Practical UC-Secure Zero-Knowledge Smart Contracts

Jayamine Alupotha[1] and Xavier Boyen[1]

Queensland University of Technology, Australia

Abstract. Zero-knowledge defines that verifier(s) learns nothing but predefined statement(s); e.g., verifiers learn nothing except the program's path for the respective transaction in a zero-knowledge contract program. *Intra-Privacy* or *insiders' zero-knowledge* — ability to maintain a secret in a multi-party computation — is an essential security property for smart contracts of Confidential Transactions (CT). Otherwise, the users have to reveal their confidential coin amounts to each other even if it is not a condition of the contract, contradicting the idea of zero-knowledge. For example, in an escrow contract, the escrow should not learn buyers' or sellers' account balances if the escrow has to pay into their accounts. Current private computational platforms, including homomorphic encryption and (ZK-)SNARK, can not be used in CT's smart contracts because homomorphic encryption requires secret key sharing, and (ZK-)SNARK requires a different setup for each computation which has to be stored on the blockchain. Existing private smart contracts are not intra-private even though they are inter-private — participants can maintain secrets from verifiers but not from other participants, accordingly. To fill this research gap, we introduce the notion of "Confidential Integer Processing" (CIP) with two *intra-private single-setup zero-knowledge programming protocols*, (1) "CIP-DLP" from the Discrete Log Problem (DLP) targeting Ring/Aggregable CT like Monero and Mimblewimble, and (2) "CIP-SIS" from Approximate (Ring-Modular-) Shortest Integer Solution Problem (Approx-SIS) aiming at lattice-based Ring/Aggregable CT. To the best of our knowledge, our CIP protocols are the *first practical public* zero-knowledge contract protocols that are also secure under the Universal Composability (UC) framework without any hardware magic or trusted offline computations.

Keywords: Zero-Knowledge · Smart Contracts · Confidential Transactions · Universal Composability.

1 Introduction

Confidential Transactions (CT) are interesting cryptographic protocols that hide senders' and receivers' coin amounts from the verifiers yet provide tools to verify that (1) coins are not stolen, and (2) hidden coins are non-negative and do not overflow, e.g., that coins are in $[0, 2^{64})$. There are two types of CTs;

- Ring-CTs that obfuscate the real sender(s) and receiver(s) with shadow participants, e.g., Monero [45, 52] and Lattice-based [1, 24, 25] and

- Aggregable CTs that allow the deletion of history (spent coin records) from the blockchain, e.g., Mimblewimble [28,33,35,53] and Lattice-based [2,57].

However, there are *no* practical smart contracts for these confidential transactions; hence the advantages of confidentiality can not be used in decentralized applications. Let us explain our claim.

What is a smart contract? A smart contract is a *Boolean program* that can be attached to a coin record like an Unspent Transaction Output (UTXO) or account. When the coin record is being spent in a transaction, blockchain verifiers execute the program by giving the transaction as the program's input. If the program outputs `TRUE`, then the verifiers accept that the transaction meets the conditions of the contract. Otherwise, or for a `FALSE`, the verifiers reject the transaction. Due to these programmable contracts, many decentralized applications can be built on top of blockchains, e.g.,

- online-shopping with secure escrow mechanisms and
- federated machine learning with fair combiners.

Traditional Contracts are not confidential. Traditional smart contracts like Solidity [27] provide advanced features for decentralized applications. However, they are not confidential, i.e., hidden coin amounts have to be revealed in order to use these contracts. Therefore, these smart contracts can not be used for CTs.

Homomorphic Encryption is not suitable for CTs' smart contracts. Homomorphic encryption (HEnc) [10,30,54] is a popular private computation mechanism. Evidently, Zkay [51] and Zeestar [50] are contract languages built on HEnc, mainly for account-based Ethereum [56]. However, in HEnc, all the confidential values should be hidden using the same (root-)encryption key, which is **not** compatible with confidential coins since each confidential coin must be protected with a random independent key. Thus, HEnc or its variants can not provide private computations for confidential transactions.

Private Circuit Computations (PCC) are not practical for smart contracts. PCCs' objective is to show that some confidential numbers satisfy a given arithmetic circuit (see Table 1 for references of PCCs). Most PCCs are based on Succinct Non-interactive ARguments of Knowledge (SNARK) protocols [31,38,42]. Typically, PCC follows these steps; (1) identify the circuit; (2) generate the setup (a.k.a. public parameters) for the circuit; (3) generate zero-knowledge proofs for the input confidential values if they satisfy the circuit; (4) verify the generated zero-knowledge proofs against the input confidential values. In theory, if the CT's confidential coins are generated as same as PCC's confidential values, PCC could be used to do private computations for CT. However, using PCC directly as smart contracts is *impractical* since each contract requires a large setup that should be stored in the blockchain, e.g., while the setup size is linear in the size of the circuit, typically, it is in Kilobytes (DLP) or Mega/Gigabytes (lattices) excluding the proof. For example, Zether [13] states that their protocol is impractical for confidential transactions due to the storage cost.

Efficient private contracts exist but need trusted components. Private smart contracts such as Hawk [39], Arbitrum [37], Ekiden [19], Zkay [51], ZEXE [9], ZoKrates [23], and ZeeStar [50] are not fully trustless since either they depend on Trusted Execution Environments (TEE) or do private computations offline using a trusted executors(s) to reduce the computational cost. While hackers have challenged the security of TEE throughout history, offline computations are also not endorsed in *decentralized systems* since the main blockchain cannot guarantee the validity of an offline computation unless its consensus verifiers *fully* verify the computation.

1.1 Our Contribution

As a solution, we propose "Confidential Integer Processing" (CIP), the first practical zero-knowledge integer programming protocol from DLP and Approx-SIS with the following properties.

1. Single-setup - All contracts use the same setup, and it is less than a Megabyte.
2. Inter-private - Participants hide secrets from the verifiers.
3. Intra-private - Participant(s) can maintain secrets from other participants of the same contract.
4. Zero-Knowledge - Verifiers learn nothing except the contract path for the given transaction.
5. Non-Interactive - No interactions between verifiers and participants.
6. Trustless - No trusted hardware or executor is used.
7. Universally Composable - Both CIP-DLP and CIP-SIS are secure according to the UC framework (this is our main technical contribution!).

Universal Composability. The framework [15, 16] guarantees strong security for multi-party composed protocols like our CIPs since the interactions between the multiple parties and multiple protocols must be thoroughly modeled to realize the security. Informally, UC says that if there is an ideal function with the desired security properties, and a real protocol that acts the same as the ideal function, then the real protocol also has the ideal function's security properties. This paper's main contributions are the followings.

– *The first ideal system for zero-knowledge programming according to the UC framework of [16]* — Achieving our security properties (zero-knowledge, knowledge soundness, and intra-privacy) is well known to be tedious in UC even for simple protocols, not only complex protocols like ours. We show how to define clear UC models easily with these properties, which can be used for future zero-knowledge protocols.
– *Practical CIP-DLP and CIP-SIS protocols that realize the UC model* — Our CIP-DLP and CIP-SIS's operations and conditionals are *novel protocols* except Bulletproof range proofs [14] used in CIP-DLP (we use [14] due to its logarithmic size while CIP-SIS uses a novel range proof protocol). Our CIP supports typical (1) integer operations: "addition", "multiplication", "signed or unsigned division", and (2) conditionals: "equal", "not equal", "less than and equal" and "greater than and equal".

Let us explain how CIPs work at a higher level.

Confidential Integers. CIP supports confidential integers denoted `cint(v, k)` where the actual integer `v` is hidden under a secret key `k`. Let g and h be generators (the DL of h to g is unknown) of a prime-order q group $\mathbb{G}$ such that the DL of g and h are unknown. In CIP-DLP, a confidential integer is a Pedersen commitment $\texttt{cint}(v, k) = g^k h^v \in \mathbb{G}$. Assume that $\mathbb{X}$ is a polynomial ring such as $\mathbb{Z}_q[X]/[X^{N+1}]$, and Approx-SIS of $(n, m, q, \gamma, \gamma', N)$ is hard for $\vec{\boldsymbol{V}} \in \mathbb{X}_q^n$ and $\vec{\boldsymbol{K}} \in \mathbb{X}_q^{n \times m-1}$ (see Definition 4). In CIP-SIS, $\texttt{cint}(v, \vec{\boldsymbol{k}} \in \mathbb{X}^{m-1})$ is $\texttt{highbits}_{\gamma'}(\vec{\boldsymbol{V}}[v, 0, ..0] + \vec{\boldsymbol{K}}\vec{\boldsymbol{k}}) \in \mathbb{X}_q^n$ when $[v, 0, ..0] \in \mathbb{X}$, and the infinity norms of v and $\vec{\boldsymbol{k}}$ are smaller than γ. Here, $\texttt{highbits}_{\gamma'}()$ drops the last $\lfloor \log_2 \gamma' \rfloor$ bits of each coefficient.

Confidential Integer Processing for Smart Contracts. CIP can be used for any CT if their confidential coins are generated the same as confidential integers or if there is a way to prove that a confidential coin's value is equal to a confidential integer's value. For example, CIP-DLP can be directly[1] used for Monero [45] and Grin-Mimblewimble [53] since they store confidential coins as Pedersen commitments. Also, confidential coins of lattice-based MatRiCT [25] and LACTx [2] can be converted into confidential integers even though the confidential coins commits binary values, not the full integer [2].

Modularity and Intra-Privacy. CIP's operations are *modular*, — a proof for each operation or conditional is computed solely based on its inputs — which is the *opposite* of creating a single proof for an arithmetic circuit. Also, CIP supports "multi-party equal" statements required for intra-privacy. As a result, two participants can show that they have the same integer without disclosing any of the secret keys. We will explain how our "multi-party equal" protocol works. Assume that two participants $\mathcal{P}_1$ and $\mathcal{P}_2$ have two DLP confidential integers $c_1 = g^{k_1} h^{v_1}$ and $c_2 = g^{k_2} h^{v_2}$, respectively. They do not want to share secret keys, but they want to show that the confidential integers hold the same integer.

- Each participant i picks a secret nonce k'_i from $[0, q)$ and computes a confidential integer of zero : $t_i = g^{k'_i} \in \mathbb{G}$. They each share t_i. Let $\boldsymbol{t} = \{t_1, t_2\}$ be the shared values.
- They both compute: $t = t_1^{\texttt{hash}(\boldsymbol{t}, t_1)} \times (t_2^{\texttt{hash}(\boldsymbol{t}, t_2)})^{-1}$.
- Then each participant i computes a challenge x by hashing c_1, c_2, t, e.g., $x = \texttt{hash}(c_1, c_2, t) \in [0, q)$ and computes $s_i = \texttt{hash}(\boldsymbol{t}, t_i) k'_i + x k_i \in [0, q)$. Then they share s_1 and s_2.
- Finally, they compute the proof $\pi = (t, s{=}(s_1 - s_2)) \in (\mathbb{G}^2, [0, q))$ for the input confidential integers (c_1, c_2). Any one of them can send π to the verifier.

We illustrate the diagram of the protocol in Figure 1. The verification is, $t \times (c_1 \times c_2^{-1})^{\texttt{hash}(c_1, c_2, t)} \overset{?}{=} g^s \in \mathbb{G}$. Then we can verify that c_1 and c_2 have the

[1] CIPs have to change public parameters according to the applications' public parameters which can be easily done.

[2] One may wonder why we do not define confidential integers similar to MatRiCT [25] and LACTx [2]. The reason is that these coins are not additively homomorphic like ours due to the binary form of v.

same integer without sharing keys (Note: these "multi-party equal" proofs are indistinguishable from "equal" proofs which will be discussed later in the paper).

Multi-Party Equality Check

$$\begin{array}{lcl}
\mathcal{P}_1 & & \mathcal{P}_2 \\
k_1' \xleftarrow{\$} [0,q) \Rightarrow t_1 = g^{k_1'} & & k_2' \xleftarrow{\$} [0,q) \Rightarrow t_2 = g^{k_2'} \\
\boldsymbol{t} = \{t_1, t_2\} & \xrightarrow{\quad t_1 \quad} \qquad \xleftarrow{\quad t_2 \quad} & \boldsymbol{t} = \{t_1, t_2\} \\
t = t_1^{\mathtt{hash}(\boldsymbol{t}, t_1)} \times (t_2^{\mathtt{hash}(\boldsymbol{t}, t_2)})^{-1} & & t = t_1^{\mathtt{hash}(\boldsymbol{t}, t_1)} \times (t_2^{\mathtt{hash}(\boldsymbol{t}, t_2)})^{-1} \\
x = \mathtt{hash}(c_1, c_2, t) & & x = \mathtt{hash}(c_1, c_2, t) \\
s_1 = \mathtt{hash}(\boldsymbol{t}, t_1) k_1' + x k_1 & \xrightarrow{\quad s_1 \quad} \qquad \xleftarrow{\quad s_2 \quad} & s_2 = \mathtt{hash}(\boldsymbol{t}, t_2) k_2' + x k_2 \\
\pi = (t, s{=}(s_1 - s_2)) & & \pi = (t, s{=}(s_1 - s_2))
\end{array}$$

Fig. 1. Multi-Party Equal Proofs such that $t \times (c_1 \times c_2^{-1})^{\mathtt{hash}(c_1, c_2, t)} \stackrel{?}{=} g^s \in \mathbb{G}$

CIP provides intra-privacy since any contract can be separated into multiple contracts due to this modularity and combined back by using "multi-party equal" statements. For example, two participants want to prove that their coin amounts are $\geq$ some `limit` and to keep `limit` hidden from the public. They:

- separately compute two confidential integers for `limit` with different keys: $\mathtt{cint}(\mathtt{limit}, \mathtt{k}_1)$ and $\mathtt{cint}(\mathtt{limit}, \mathtt{k}_2)$;
- secretly compute "larger than and equal" proofs for their coin amounts;
- together create a "multi-party equal" proof to show that $\mathtt{cint}(\mathtt{limit}, \mathtt{k}_1)$ and $\mathtt{cint}(\mathtt{limit}, \mathtt{k}_2)$ have the same integer value without revealing `k1` and `k2` to each other; and
- combine all three proofs into a single proof which is *the zero-knowledge proof for the contract.*

In that way, participants **do not** learn others' coin amounts but ensure that both participants used the same `limit`. We can use "multi-party equal" statements to achieve intra-privacy at a minimal cost since these "multi-party equal" proofs are relatively smaller (even in Approx-SIS, it is only a hash challenge and a short polynomial vector of $\mathbb{X}^{m-1}$).

Why do not we use a common confidential integer for `limit`*?* CIPs' operations and conditionals are modular, and some operations like multiplication and division are inspired by *sigma protocols* with Fiat-Shamir challenges [26, 49]. Hence, even knowing a secret key of one input is sufficient to identify the real value or characteristics of other inputs, which is a violation of the zero-knowledge argument, e.g., using a common confidential integer for `limit` may reveal hidden integers of other confidential integers. CIP solves this problem via "multi-party equal" statements which do not require any secret key sharing.

Modularity and Running-Time Verification Cost. Our CIPs also provide "output as soon as possible" property, unlike arithmetic circuit computations where no outputs are returned until the end of the program. Therefore, a contract can be stopped if it has early `return`s, and the verifiers do not need to run the entire circuit. Another advantage is that the cost of computation can be easily calculated by weighing each modular operation at *running time*, i.e., the cost will be only computed for that particular path, not for the whole contract if the program terminates early.

Escrow Mechanisms are the most common contract type in blockchains. The escrow holds a buyer's coins until either the seller sends the goods to claim the coins or refunds the buyer while taking a small commission. First, the buyer sends coins to the escrow with a contract stating the buyer's and seller's account addresses (mostly public keys) and the commission. The escrow can keep the commission and sends (coins - commission) either to the buyer or the seller. However, the escrow is prohibited from spending those (coins - commission) anywhere else. When the contract protocol is not intra-private, the seller or the buyer has to reveal their input account balances to the escrow to compute the proof for the contract. A naive solution would be to create an account for each contract, but it overloads the blockchain with unnecessary data. Our CIPs solve this privacy problem easily, i.e., whoever (seller/buyer) receives the coins shows that (coins - commission) is equal to received coins or (output account balance - input account balance) by computing a "multi-party equal" statement with the escrow. Therefore, the users do not have to create multiple accounts to hide their coin amounts, and the blockchain will not be unnecessarily overloaded.

Federated Learning. CIP can be used for other multi-party applications outside blockchains, e.g., *federated learning* [41] since CIPs are generic programming languages. Federated learning introduces a multi-party learning model with the freedom to hide individuals' raw data. Each individual trains their raw data into a partial model, and then a trusted combiner aggregates those partial models into a single trained model. However, these unverifiable combiner/individuals can be malicious, i.e., a malicious combiner can aggregate corrupted partial models, to get a biased trained model, e.g., for market manipulation. A CIP program can mitigate this by setting accepted ranges [7, 36] to data without showing individual raw data.

Other Applications. Also, CIP can be used for *outsourced-security management software* like URL defense [55] to protect the privacy of users without disrupting the functionalities of the software, e.g., "not equal" proofs for blacklisted URLs. *Verifiable private computations on cloud data* equally benefit from CIP since users do not have to reveal original data but the characteristics of their data. Many more applications are possible, including more privacy-friendly verification for Distributed Apps (DApps) and Decentralized Finance (DeFi).

Related Work. We summarize the previously discussed related work; HEnc, PCC, and previous private smart contract protocols in Table 1. Here, "Trustless" means it provides

- *public verification* - anyone can verify proofs that is opposite to a designated verifier who requires some secret knowledge, and
- *transparency* - no trusted setup is needed to create a Common Reference String (CRS), nor is there any language-dependent CRS or *hardware magic.*

From Table 1, most of the existing smart contracts (except Zether [13]) require some trusted components like off-line executions or trusted hardware.

Work	Type	Trustless	Inter-private	Intra-private	Quantum-Safe	Ring-CT	Aggregable-CT	UC Safe
[10,30,54]	Homomorphic Encryption	✓	-	-	✓	-	-	-
Scripts [47]	Fixed Scripts	✓	✓	-	✗	-	✓	✗
Ligero [3]	Computations	✓	✓	✗	✓	-	-	-
Fractal [20]	Computations	✓	✓	✗	✓	-	-	-
Aurora [5]	Computations	✓	✓	✗	✓	-	-	-
zk-STARK [4]	Computations	✓	✓	✗	✓	-	-	-
Gennaro [29]	Computations	✗	✓	✗	✓	-	-	-
Nitulescu [44]	Computations	✗	✓	✗	✓	-	-	-
Boschini [8]	Computations	✓	✓	✗	✓	-	-	-
Ishai et al. [34]	Computations	✗	✓	✗	✓	-	-	-
Zilch [43]	Computations	✓	✓	✗	✓	-	-	-
Enigma [58]	Contracts	✗	✓	✗	✗	✗	✗	✗
Hawk [39]	Contracts	✗	✓	✗	✗	✗	✗	✗
Arbitrum [37]	Contracts	✗	✓	✗	✗	✗	✗	✗
Ekiden [19]	Contracts	✗	✓	✗	✗	✗	✗	✗
Zether [12]	Contracts	✓	✓	✗	✗	✗	✗	✗
ZEXE [9]	Contracts	✗	✓	✗	✗	✗	✗	✗
ZoKrates [23]	Contracts	✗	✓	✗	✗	✗	✗	✗
ZKay [51]	Contracts	✗	✓	✗	✗	✗	✗	✗
SodsMPC [22]	Contracts	✗	✓	✗	✓	✗	✗	✗
ZeeStar [50]	Contracts	✗	✓	✗	✗	✗	✗	✗
Our CIP-DLP	Contracts	✓	✓	✓	✗	✓	✓	✓
Our CIP-SIS	Contracts	✓	✓	✓	✓	✓	✓	✓

Table 1. Comparison of Private Computational Platforms, Homomorphic Encryption, and Private Smart Contracts ("Intra-privacy" means "Insiders' Zero-Knowledge").

Private computations that are plausibly suitable as contracts are not actually intra-private or single-setup protocols. Script-less scripts [47] proposed for Mimblewimble [53] are not generic and only do addition-based operations. HEnc-based computations are inherently insecure for confidential transactions since confidential coins must have independent keys. Therefore, we claim CIP-DLP and CIP-SIS are the first practical generic programming languages for Ring and aggregable CTs that are also trustless and intra-private.

Also, for the interest of cryptographers, we formally define insiders' zero-knowledge, or intra-privacy, and we show how to build UC models easily for complex zero-knowledge protocols by expanding the discussion started in [17].

Road-map. First, we define the preliminaries of the paper, including the UC framework in Section 2. Then we explain our UC model and the expected security properties of CIP in Section 3. Operations and conditionals of CIP-DLP and CIP-SIS are stated in Section 4, and their UC security is described in Section 5. **Note** that readers unfamiliar with the UC framework can find nicely explained simple examples in [16, 17, 21] or directly go to the real protocols (Section 4) that are meaningful without their ideal functions.

2 Preliminaries

Notation. $\wedge$ is the "and" operation and $\vee$ is the "or" operation. $|\cdot|$ returns the number of elements or rows in a array or vector.

For a cyclic group $\mathbb{G} = \langle g \rangle$, g denotes a generator of the group $\mathbb{G}$. $\mathbb{Z}_q^+ = \mathbb{Z}/q\mathbb{Z}$ is a ring of modular integers with representations assumed in the range $[0, q-1]$ for a modulus q. Similarly, $\mathbb{Z}_q = \mathbb{Z}/q\mathbb{Z}$ is a ring of modular integers in the range $[-\frac{q-1}{2}, \frac{q-1}{2}]$ for an odd q. For an even q, the range is $(-\frac{q}{2}, \frac{q}{2}]$. $\mathbb{X}$ is a polynomial ring of $\mathbb{Z}[X]/[X^N + 1]$ with degree $N = 2^k$ for some integer $k > 0$. $\mathbb{X}_q$ is a fully splitting ring of $\mathbb{Z}_q[X]/[X^N + 1]$ for a prime q such that $q \equiv 1 \pmod{2N}$. A polynomial $a_{N-1}X^{N-1} + .. + a_1 X + a_0 \in \mathbb{X}_q$ is denoted as $\vec{a} = [a_0, ..a_{N-1}]$ when each coefficient is in $\mathbb{Z}_q$. From here on, we assume q to be prime.

Simple bold letters like $\boldsymbol{a}$ denotes integer vectors. Similarly, $\vec{\boldsymbol{a}}$ denotes a vector of polynomials, and $\vec{\boldsymbol{A}}$ denotes a matrix of polynomials. The ith element of a vector $\boldsymbol{a}$ is a_i or we can denote the entire vector as $\boldsymbol{a} = [a_0, .., a_{n-1}] = [a_i]_{i=0}^{n-1}$ when $\boldsymbol{a}$ has n elements. Also, $\boldsymbol{A} = [\boldsymbol{A}_0, .., \boldsymbol{A}_{m-1}]$ when $\boldsymbol{A}$ has m vectors. $\vec{a}\vec{b}$ denotes the polynomial multiplication of $\vec{a}$ and $\vec{b}$. Also, $\vec{a}\vec{a} = \vec{a}^2$. When x is an integer, $x\boldsymbol{a}$ denotes that every element in $\boldsymbol{a}$ is multiplied by x. $\vec{x}\vec{\boldsymbol{a}}$ denotes that every polynomial in $\vec{\boldsymbol{a}}$ is multiplied by $\vec{x}$. We use "+", "−", and "∘" for element-wise or polynomial-wise addition, subtraction, and multiplication respectively, e.g., $\vec{\boldsymbol{a}} \circ \vec{\boldsymbol{b}} = [\vec{a}_0\vec{b}_0, .., \vec{a}_{n-1}\vec{b}_{n-1}]$. The norms are defined as follows, $\|\vec{a}\| = \mathtt{max}([|a_i|]_{i=0}^{N-1})$ and $\|\vec{a}\|_1 = \sum_{i=0}^{N-1} |a_i|$. For a vector of polynomials $\vec{\boldsymbol{a}}$, $\|\vec{\boldsymbol{a}}\| = \mathtt{max}([\|\vec{a}_i\|]_{i=0}^{n-1})$. Note that $\mathbf{0}^n \in \mathbb{Z}_q^n$ is a zero vector and $\vec{0} \in \mathbb{X}_q$ is the polynomial with all zero coefficients. $\mathtt{rot}(val, i \in [0, N-1])$ is a polynomial whose ith coefficient is val, and all other coefficients are zeros. $\mathtt{bin}(v) = \vec{b} \in \mathbb{X}_q$ such that $v = \sum_{i=0}^{N-1} 2^i b_i$. $\mathtt{lowbits}_p(v)$ keeps the $\lfloor \log_2 p \rfloor$ low-bits of v, and $\mathtt{highbits}_p(v)$ is $v - \mathtt{lowbits}_p(v)$. Also, $\mathtt{up}_p(v) = v \cdot 2^{\lfloor \log_2 p \rfloor}$. We use the above-mentioned functions for polynomials vectors as well. $m \xleftarrow{\$} \mathcal{M}$ denotes that m is drawn uniformly at random from a (finite) set $\mathcal{M}$. We use λ as the security parameter throughout the paper unless otherwise mentioned. $\epsilon(\lambda) = 1/o(\lambda^c)$ is a negligible function which vanishes faster than any polynomial of degree c, $\forall c \in \mathbb{N}$. We use pp to denote public parameters, $\mathcal{A}$ for real adversaries, and $\mathcal{S}$ for simulated adversaries in a formal security reduction.

Throughout the paper, we casually use "hiding" or "indistinguishability" to mean computational hiding unless otherwise mentioned.

Definition 1 (Computational Hiding). *Two distributions generated from some public parameters pp_λ; $D_0(pp_\lambda)$ and $D_1(pp_\lambda)$ are indistinguishable or $D_0 \approx D_1$ if for any p.p.t. distinguisher $\mathcal{A}$,*

$$\mathtt{Adv}_{\mathcal{A}}^{\mathtt{HID}} = 2 \left| \frac{1}{2} - Pr\left[i \stackrel{?}{=} j \;\middle|\, i \xleftarrow{\$} [0,1]; k \xleftarrow{\$} D_i; j \leftarrow \mathcal{A}(pp_\lambda, k) \right] \right| \leq \epsilon(\lambda)$$

Definition 2 (Discrete Log Problem). *For $\mathbb{G} = \langle g \rangle$ of prime order q, $\mathtt{Adv}_{\mathbb{G}}^{DL}$ for an adversary $\mathcal{A}$ is defined as, $\mathtt{Adv}_{\mathbb{G},\mathcal{A}}^{DL} := Pr[y \stackrel{?}{=} g^x |\ y \xleftarrow{\$} \mathbb{G}, x \xleftarrow{\$} \mathcal{A}(y)]$. The DL problem is (τ, e)-hard if $\mathcal{A}(\tau, e)$ runs it at most τ times and $\mathtt{Adv}_{\mathbb{G},\mathcal{A}}^{DL} \leq e$.*

Definition 3 (Inhomogeneous Module Short Integer Solution Problem (iMSIS)). *The advantage of an adversary $\mathcal{A}$ solving an special instance of iMSIS of $pp = (n, m, q, \gamma, N)$ after one execution is given by,*

$$\mathtt{Adv}^{\mathtt{iMSIS}}_{pp,\mathcal{A}} = Pr\left[\|\vec{\boldsymbol{s}}\| \leq \gamma \wedge \vec{\boldsymbol{H}}\vec{\boldsymbol{s}} = \vec{\boldsymbol{y}} \in \mathbb{X}_q^n \,\middle|\, \begin{array}{r} \vec{\boldsymbol{H}} \xleftarrow{\$} \mathbb{X}_q^{n\times m}; \vec{\boldsymbol{y}} \xleftarrow{\$} \mathbb{X}_q^n \\ (\vec{\boldsymbol{s}} \in \mathbb{X}^m) \leftarrow \mathcal{A}(pp_{\lambda,L}, \vec{\boldsymbol{H}}, \vec{\boldsymbol{y}}) \end{array} \right]$$

Definition 4 (Approximate Inhomogeneous Module Short Integer Solution Problem (Approx-SIS)). *The advantage of an algorithm $\mathcal{A}$ solving Approx-SIS of $pp = (n, m, q, \gamma, \gamma', N)$ after one execution is given by,*

$$\mathtt{Adv}^{\mathtt{Approx-SIS}}_{pp,\mathcal{A}} = Pr\left[\|\vec{\boldsymbol{s}}\| \leq \gamma \wedge \|\vec{\boldsymbol{H}}\vec{\boldsymbol{s}} - \vec{\boldsymbol{y}} \in \mathbb{X}_q^n\| \leq \gamma' \,\middle|\, \begin{array}{r} \vec{\boldsymbol{H}} \xleftarrow{\$} \mathbb{X}_q^{n\times m}; \vec{\boldsymbol{y}} \xleftarrow{\$} \mathbb{X}_q^n \\ (\vec{\boldsymbol{s}} \in \mathbb{X}^m) \leftarrow \mathcal{A}(pp_{\lambda,L}, \vec{\boldsymbol{H}}, \vec{\boldsymbol{y}}) \end{array} \right]$$

Theorem 1. *From [18], Approx-SIS and iMSIS are tightly bound such that* $\mathtt{Adv}^{\mathtt{iMSIS}}_{(n,n+m,q,\gamma,N)} \geq \mathtt{Adv}^{\mathtt{Approx-SIS}}_{(n,m,q,\gamma,\gamma+\gamma',N)}$ *and* $\mathtt{Adv}^{\mathtt{iMSIS}}_{(n,n+m,q,\gamma+\gamma',N)} \leq \mathtt{Adv}^{\mathtt{Approx-SIS}}_{(n,m,q,\gamma,\gamma',N)}$.

Hints. For constructions based on Approx-SIS, only the higher bits are taken for computations. Hence, large errors can occur during the polynomial multiplications. To fix those errors, we use hints similar to [48]. These hint polynomials are $\{-1, 0, 1\}$ polynomial vectors that hold at most χ number of ±1. Creating and using hint polynomials are stated below.

```
$\underline{\mathtt{hints}(\chi, \vec{\boldsymbol{a}} \in \mathbb{X}_q^n, \vec{\boldsymbol{b}} \in \mathbb{X}_q^n)}$:
$\vec{\boldsymbol{h}} = \vec{\boldsymbol{a}} - \vec{\boldsymbol{b}} \in \mathbb{X}_q^n$
if $\|\vec{\boldsymbol{h}}\| > 1 \wedge \|\vec{\boldsymbol{h}}\|_1 > \chi$: return $\perp$
return $\vec{\boldsymbol{h}}$
```

```
$\underline{\mathtt{use\_hints}(\chi, \vec{\boldsymbol{a}} \in \mathbb{X}_q^n, \vec{\boldsymbol{h}} \in \mathbb{X}_q^n)}$:
if $\|\vec{\boldsymbol{h}}\| > 1 \wedge \|\vec{\boldsymbol{h}}\|_1 > \chi$: return $\perp$
return $\vec{\boldsymbol{b}} = \vec{\boldsymbol{a}} - \vec{\boldsymbol{h}} \in \mathbb{X}_q^n$
```

Universal Composability - Hybrid Model

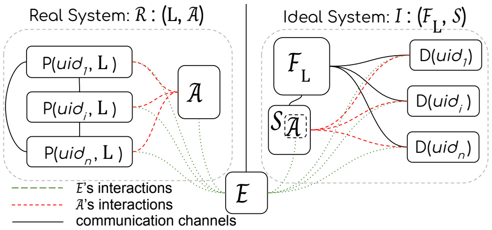

Fig. 2. Interactions of UC-Framework

The Universal Composability (UC) model [15–17, 21] is a framework used to analyze the security of composed protocols that combine multiple protocols together. In the UC model, there are an ideal system, a real system, and an environment $\mathcal{E}$ (challengee). $\mathcal{E}$'s task is to identify which system it is interacting with; ideal or real (the way $\mathcal{E}$ interacts with systems is discussed in the latter part of the section). We say the real system realizes the ideal system if $\mathcal{E}$ can not identify the correct system with more than $1/2 + \epsilon(\lambda)$ probability. In that case, informally, Universal Composability says that the real system provides the security properties of the ideal system. Formally, $\mathcal{E}$ executes a program on **interactive Turing Machines** (iTMs), which have multi-tapes for inputs and outputs. "Image" is the state of all iTMs' tapes at any step of the program which is subjected to change throughout the program. $\mathcal{E}$ can stop the program and read the image at any time. The real system $\mathcal{R} : (\mathtt{L}, \mathcal{A})$ of real participants

$[\mathcal{P}(\mathtt{uid}_i, \mathtt{L})]^*_{i=0}$ consists of the protocol $\mathtt{L}$ and the real adversary $\mathcal{A}$. Similarly, the ideal system $\mathcal{I} : (\mathcal{F}_{\mathtt{L}}, \mathcal{S}_{\mathcal{A}})$ of dummy participants $[\mathcal{D}(\mathtt{uid}_i)_i]^*_{i=0}$ has an ideal function $\mathcal{F}_{\mathtt{L}}$ (it is trusted, has infinite computational power, and can serve infinite numbers of participants at the same time) and a simulated adversary $\mathcal{S}_{\mathcal{A}}$. In the UC model, real or dummy participants are the iTMs whose input tapes can be written by $\mathcal{E}$. Also, $\mathcal{E}$ can instruct $\mathcal{A}$ to attack the system like corrupting real participants, eavesdropping a communication between two real participants, or altering a message that is being sent from a participant to another.

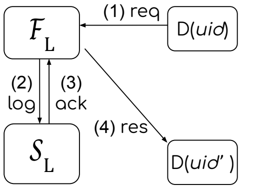

Fig. 3. Single Participant Protocol

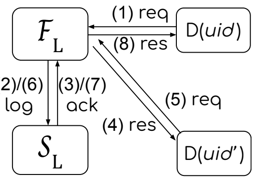

Fig. 4. Multi-Party Computations

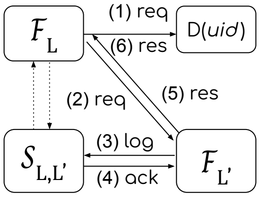
$\mathcal{F}_{\mathrm{L}}$
(1) req
D(uid)
(6) res
(2) req
(5) res
$\mathcal{S}_{\mathrm{L,L'}}$
(3) log
(4) ack
$\mathcal{F}_{\mathrm{L'}}$

Fig. 5. Composed Protocol Computations

The ideal system is modelled as follows to simulate what is happening in the real system. Initially, all participants are `honest`. If $\mathcal{A}$ is instructed to corrupt $\mathcal{P}(\mathtt{uid}_i, \mathtt{L})$ by $\mathcal{E}$ then $\mathcal{S}_{\mathcal{A}}$ corrupts $\mathcal{D}(\mathtt{uid}_i)_i$ at the same time and saves $\mathcal{D}(\mathtt{uid}_i)_i$'s state as `corrupt`. There are three types of communication in ideal systems.

The first type is "Single-Participant Protocol" (see Figure 4) which only involves a single participant and a protocol. If $\mathcal{E}$ writes into $\mathcal{P}(\mathtt{uid})$'s input tape instructing to execute a sub-function of $\mathtt{L}$, in the ideal system:

- $\mathcal{D}(\mathtt{uid})$ sends a request (`req`) to $\mathcal{F}_{\mathtt{L}}$ asking to execute the sub-function,
- $\mathcal{F}_{\mathtt{L}}$ sends a backdoor request (`log`) to $\mathcal{S}_{\mathcal{A}}$ with `req`'s data,
- $\mathcal{S}_{\mathcal{A}}$ executes the sub-function according to `req`'s data and sends acknowledgement (`ack`) to $\mathcal{F}_{\mathtt{L}}$ which may include the sub-function's output and $\mathcal{D}(\mathtt{uid})$'s state: `honest` or `corrupt`,
- $\mathcal{F}_{\mathtt{L}}$ sends the response (`res`) according to `ack` which will be written into $\mathcal{D}(\mathtt{uid})$'s output tape.

The second communication type is "Multi-party Computations". When $\mathcal{E}$ writes into $\mathcal{P}(\mathtt{uid}_i)$'s input tape ordering a multi-party computation with $\mathcal{P}(\mathtt{uid}')$, $\mathcal{F}_{\mathtt{L}}$, follows the steps in Figure 4, and both $\mathcal{D}(\mathtt{uid})$'s and $\mathcal{D}(\mathtt{uid})$'s tapes will be updated. Note that prior to any response, $\mathcal{F}_{\mathtt{L}}$ communicates with $\mathcal{S}_{\mathcal{A}}$ to identify whether $\mathcal{A}$ is corrupting/eavesdropping the communication. For example, $\mathcal{S}_{\mathcal{A}}$ alters (3)`ack` and (7)`ack` accordingly if $\mathcal{A}$ corrupts the communication channel.

The final communication type is "Composed Protocol Computations" which requires multiple protocols a.k.a. a hybrid model. In the hybrid model, $\mathcal{F}_{\mathtt{L}}$ and $\mathcal{F}_{\mathtt{L}'}$ share the same simulated adversary which is denoted as $\mathcal{S}_{\mathtt{L},\mathtt{L}'}$. As shown in Figure 5, $\mathcal{F}_{\mathtt{L}}$ communicates with $\mathcal{F}_{\mathtt{L}'}$ when $\mathcal{E}$ forces a participant to execute a sub-function of $\mathtt{L}$ which requires a sub-function(s) of $\mathtt{L}'$.

Universal composability says that the real system provides the same security properties as the ideal system if $\mathcal{E}$ with any p.p.t. adversary $\mathcal{A}$ can not distinguish whether it is interacting with the ideal or real system by reading the images of iTMs for any execution. We recall the formal definition from [16].

Theorem 2. *For a real system* $\mathcal{R} : (\mathtt{L}, \mathcal{A})$ *and an ideal system* $\mathcal{I} : (\mathcal{F}_{\mathtt{L}}, \mathcal{S}_{\mathcal{A}})$*,* $\mathtt{L}$ ***uc-realizes*** $\mathcal{F}_{\mathtt{L}}$ *if and only if no environment* $\mathcal{E}$ *can tell whether it is interacting with* $\mathcal{R}$ *or* $\mathcal{I}$ *more than with negligible probability.* $\mathtt{Execute}(\mathcal{E}, \mathcal{R} : (\mathtt{L}, \mathcal{A})) \approx \mathtt{Execute}(\mathcal{E}, \mathcal{I} : (\mathcal{F}_{\mathtt{L}}, \mathcal{S}_{\mathcal{A}}))$ *meaning that after each step* i *of the program, the images of* $\mathcal{I}$ *and* $\mathcal{R}$ *are the same, i.e.,* $\forall i : \left[\mathsf{Image}^{i}_{\mathcal{R},\mathcal{E},\mathcal{A}} \rightarrow \mathsf{Image}^{i+1}_{\mathcal{R},\mathcal{E},\mathcal{A}}\right] \approx \left[\mathsf{Image}^{i}_{\mathcal{I},\mathcal{E},\mathcal{S}_{\mathcal{A}}} \rightarrow \mathsf{Image}^{i+1}_{\mathcal{I},\mathcal{E},\mathcal{S}_{\mathcal{A}}}\right]$.

We emphasize that we follow the *original UC framework* of [16] where $\mathcal{S}_{\mathcal{A}}$ is stateful — $\mathcal{S}_{\mathcal{A}}$ keeps a log for each dummy party. Thus, a corruption of a real participant can be simulated exactly the same in the ideal system by sharing past states of the respective dummy party, not only the future states.

3 Confidential Integer Processing

A confidential integer is an additively homomorphic commitment of an integer and a secret key. Let `CIP` be a generic commitment scheme. Here, "additively homomorphic" means, $\mathtt{cint}(\mathtt{v}, \mathtt{k}) \pm_{\mathtt{c}} \mathtt{cint}(\mathtt{v}', \mathtt{k}') = \mathtt{cint}(\mathtt{v} \pm \mathtt{v}' \pmod{\mathtt{q}}, \mathtt{k} \pm_{\mathtt{k}} \mathtt{k}')$. We use $\pm_{c/k}$ to denote the homomorphic operations casually, e.g., $\pm_c$ becomes multiplication in DLP and modular polynomial vector addition in Approx-SIS, and $\pm_k$ is the typical addition in DLP and polynomial addition in Approx-SIS.

Let `CIP` be a generic "Confidential Integer Processing" protocol with predefined public parameters $pp_{\lambda,L}$ where L denotes the recommended bit range for integers, e.g., CIP-SIS's public parameters will be computed for integers in $(-2^L, 2^L)$ when $q \gg 2^{2L+5}$. CIP-DLP's secure range is $[-2^L, 2^L)$ such that $q > 2^{L+5}$. We only check the range during confidential integer generation but provide range proofs up to $(2L + 5)$-bits if users want to prevent overflowing.

`CIP`'s object is to prove that a set of confidential integers holds a relation r of the language $\mathcal{L}$. For example, $r([v_i]^{*}_{i=0}) = 1/0$ indicates whether the integers hold the relation or not. We can categorize these relations into either (1) modular relations or primitive relations like range, addition, multiplication, or (2) composed relations or sequentially combined modular relations. For example, the relation of range is $\mathtt{range}_L(v) = (v \overset{?}{\in} (-2^L, 2^L))$, the relation of multiplication is $\mathtt{mul}([v_i]^{*}_{i=0}) = (v_0 \overset{?}{=} \prod^{*}_{i=1} v_i) \pmod{q}$, and the equality's relation is $\mathtt{eq}(v_0, v_1) = (v_0 \overset{?}{=} v_1)$. An exemplary composed relation is $\{\mathtt{mul}([v_i]^{3}_{i=0}) \wedge \mathtt{range}_L(v_0)\}$. `CIP` provides proving and verification functions for common modular relations, e.g., $\mathtt{prove}(\mathtt{eq}, c_0, v_0, k_0, c_1, v_1, k_1)$ gets the confidential integers and their openings and outputs π. Then $\mathtt{ver}(\mathtt{eq}, c_0, c_1, \pi)$ takes the same confidential integers and π and outputs 1 (or 0). A composed relation also can be seen as a sequential proving and verification functions of modular relations, e.g., a single party `CIP` program can be

$$\begin{Bmatrix} \mathtt{mul}([v_i]^{2}_{i=0}) \\ \wedge \mathtt{eq}(v_0, v_3) \end{Bmatrix} \Rightarrow \begin{Bmatrix} \mathtt{prove}(\mathtt{mul}, [c_i, v_i, k_i]^{2}_{i=0}) \rightarrow \pi_0 \\ \mathtt{prove}(\mathtt{eq}, c_0, v_0, k_0, c_3, v_3, k_3) \rightarrow \pi_1 \end{Bmatrix} \Rightarrow \begin{Bmatrix} \mathtt{ver}(\mathtt{mul}, [c_i]^{2}_{i=0}, \pi_0) \\ \wedge \mathtt{ver}(\mathtt{eq}, c_0, c_3, \pi_1) \end{Bmatrix}.$$

However, `CIP` is a multi-party protocol, i.e., multiple participants own confidential integers of these raw integers, and they may not exchange the openings with others. For example, a participant $\mathcal{P}_2$ who has (k_3) does not want to share k_3 with $\mathcal{P}_1$ but both want to compute the equality proof π_1. In that case, we can use "multi-party equal" proofs rather than "equal". As we explained in the introduction, a "multi-party equal" can be seen as a protocol where provers keep computing individual public inputs, a.k.a. transcripts and combining them into partial proofs until they come up with the final proof. Similarly, we can generalize proving functions of complex composed relations as follows.

The initial partial proof $\pi_0 = [c_i]_{i=0}^{*}$ is a list of input confidential integers of the initial transcript set $\boldsymbol{\theta}_0 = \{c_i\}_{i=0}^{*}$. Each prover j computes his/her transcript $\theta_{t,j}$ by taking previous partial proof π_{t-1} and secret knowledge $\boldsymbol{k}_{t-1,j}$ that he/she used to generate $\theta_{t-1,j}$, e.g., the secret key of c_i for the first transcript. More specifically, we can denote the (t)th transcript generation as follows,

$$(\boldsymbol{k}_{t,j}, \theta_{t,j}) \leftarrow \texttt{prove}_t(r, \pi_{t-1}, \boldsymbol{k}_{t-1,j}).$$

Each partial proof π_t is generated according to π_{t-1} and input transcripts of participants $\boldsymbol{\theta}_t$ such that $\pi_t \leftarrow \texttt{combine}_t(r, \pi_{t-1}, \boldsymbol{\theta}_t)$. The final partial proof $\pi_{\texttt{final}}$ is the relation proof π. For example, we can rewrite the composed proving function in Figure 1 as follows,

$$\left\{\begin{array}{r}
\texttt{prove}(\texttt{mul}, [c_i, v_i, k_i]_{i=0}^{2}) \to \pi_0 \\
\forall j \in \{0,3\} : \texttt{prove}_1(\texttt{eq}, \pi_{1,0}{=}[c_0, c_3], \boldsymbol{k}_{0,j}{=}\{v_j, k_j\}) \to (\boldsymbol{k}_{1,j}{=}(k_j, k_j'), \theta_{1,j}{=}t_j \in \mathbb{G}) \\
\texttt{combine}_1(\texttt{eq}, \pi_{1,0}, \boldsymbol{\theta}_1) \to \pi_{1,1} = [c_0, c_3, t, x] \\
\forall j \in \{0,3\} : \texttt{prove}_2(\texttt{eq}, \pi_{1,1}, \boldsymbol{k}_{1,j}) \to (k_{2,j} = (k_j, k_j'), \theta_{2,j}{=}s_j) \\
\texttt{combine}_2(\texttt{eq}, \pi_{1,1}, \boldsymbol{\theta}_2) \to \pi_{1,2} = (s, t) \to \pi_1
\end{array}\right\}$$

and the composed verification process will be the same since "multi-party equal" verification is equivalent to "equal" verification. More specifically, `CIP` supports the following functionalities:

1. `CIP.setup`$()$: return $pp_{\lambda,L}$ ▷ Reading public parameters
2. `CIP.cint`(v) : return (k, c) ▷ Commitment c of (v, k) when $v \in (-2^L, 2^L)$
3. `CIP.open`(c, v, k) : return 1/0 ▷ Opening or decommitment
4. `CIP.prove`$_t(r \in \mathcal{L}; \pi_{t-1}, \boldsymbol{k}_{t-1})$: return $(\boldsymbol{k}_t, \theta_t)$ ▷ The (t)th public transcript θ_t and its secret knowledge $\boldsymbol{k}_t$ when the $(t-1)$th partial zero-knowledge argument is π_{t-1}, and the secret knowledge used in the $(t-1)$th public transcript is $\boldsymbol{k}_{t-1}$.
5. `CIP.combine`$_t(r \in \mathcal{L}; \pi_{t-1}, \boldsymbol{\theta}_t)$: return π_t ▷ A partial zero-knowledge argument π_t when users' public input transcripts are $\boldsymbol{\theta}_t$ and the $(t-1)$th partial zero-knowledge argument is π_{t-1}. Note that the initial partial argument is $\pi_0 = [c_i]_{i=0}^{*}$ for input confidential integers, and the final partial argument $\pi_{\texttt{final}}$ is the proof the relation $r \in \mathcal{L}$
6. `CIP.ver`$(r \in \mathcal{L}; \pi_0 = [c_i]_{i=0}^{*}, \pi)$: return 1/0 ▷ Verification of a final zero-knowledge argument π for a given set of confidential integers $[c_i]_{i=0}^{*}$

3.1 Security Properties

Our target is to build binding confidential integers (Definition 6), and zero-knowledge arguments (Definition 8) that are also secure under simulation extractability (Definition 7). Let us explain these expected security definitions.

Completeness of CIP states that honestly generated confidential integers can be opened against the same openings, and honestly generated arguments are always considered valid for all relations in the language $\mathcal{L}$.

Definition 5 (Completeness). $\texttt{CIP}$ *is complete for any p.p.t. adversary* $\mathcal{A}$ *and* $pp_{\lambda,L} \leftarrow \texttt{CIP.Set}()$ *if,*

$$Pr\left[\texttt{CIP.open}(c, v, k) \middle| \; v \xleftarrow{\$} (-2^L, 2^L); (k, c) := \texttt{CIP.cint}(v)\right] \geq 1 - \epsilon(\lambda)$$

$$Pr\left[\texttt{CIP.ver}(r, [c_i]^*_{i=0}, \pi) \middle| \begin{array}{r} r \xleftarrow{\$} \mathcal{L}; [v_i]^*_{i=0} \xleftarrow{\$} (-2^L, 2^L) \; s.t. \; r([v_i]^*_{i=0}) = 1 \\ [(k_i, c_i) := \texttt{CIP.cint}(v_i)]^*_{i=0} \\ \pi := \texttt{CIP.prove}(r, [c_i]^*_{i=0}, [v_i, k_i]^*_{i=0}) \end{array}\right] \geq 1 - \epsilon(\lambda).$$

Binding means that no p.p.t. adversary can find two different openings; the integer and the key, for the same confidential integer. Hence, a different opening can not be claimed after the publication of the confidential integer.

Definition 6 (Binding). *When* $pp_{\lambda,L} \leftarrow \texttt{CIP.Set}()$*,* $\texttt{CIP}$ *is binding for any p.p.t. adversary* $\mathcal{A}$ *if* $\texttt{Adv}^{\texttt{BID}}_{\texttt{CIP},\mathcal{A}}$ *is ,*

$$Pr\left[\begin{array}{r} (v_0, k_0) \overset{?}{\neq} (v_1, k_1) \wedge v_0, v_1 \in (-2^L, 2^L) \\ \wedge \texttt{CIP.open}(c, v_0, k_0) \wedge \texttt{CIP.open}(c, v_1, k_1) \end{array} \middle| \begin{array}{r} \mathcal{A}(pp_{\lambda,L}) \rightarrow (c, v_0, k_0, \\ v_1, k_1) \end{array}\right] \leq \epsilon(\lambda).$$

Simulation Extractability is a stronger notion of Knowledge Soundness (KS), which says that the probability of finding a valid argument for a set of confidential integers that do not hold the relation is negligible. Hence it can not be checked in the real world; general security models use a simulation extractor who can extract witnesses given the argument and the commitments in a simulated world, e.g., public parameters or language-dependent CRSs with trapdoors or general Forking Lemma with rewinding adversaries [11]. If CIP is simulation extractable, the probability of any p.p.t. adversary $\mathcal{A}$ finding a valid argument π for a relation r and a set of confidential integers such that the simulated extractor can not extract a set of witnesses that holds r from the confidential integers is negligible. Considering DLP and Approx-SIS problems, we define a simulator $\mathcal{K}(pp_{\lambda,L}, \mathcal{T}, r \in \mathcal{L}; [c_i]^*_{i=0}, \pi)$ who can extract values of $[c_i]^*_{i=0}$ using π of $r \in \mathcal{L}$ if it is given a trapdoor $\mathcal{T}$ for $pp_{\lambda,L}$. We define simulation extractability as follows.

Definition 7 (Simulation Extractability). *Let* $pp_{\lambda,L} \leftarrow \texttt{CIP.Set}()$*, and* $\mathcal{T}$ *be a trapdoor of* $pp_{\lambda,L}$*. Here,* $r(\cdot) \in \mathcal{L}$ *is a relation.* $\texttt{CIP}$ *is simulation extractable for any p.p.t.* $\mathcal{A}$ *if,*

$$\texttt{Adv}^{\texttt{KS}}_{\texttt{CIP},\mathcal{A}} = Pr\left[\begin{array}{r} r \overset{?}{\in} \mathcal{L} \wedge \texttt{CIP.ver}(r; [c_i]^*_{i=0}, \pi) \\ \wedge \neg r(\mathcal{K}(pp_{\lambda,L}, \mathcal{T}, r; [c_i]^*_{i=0}, \pi)) \end{array} \middle| \; \mathcal{A}(pp_{\lambda,L}) \rightarrow (r; [c_i]^*_{i=0}, \pi)\right] \leq \epsilon(\lambda).$$

Zero-Knowledge Argument (Computational) implies that a p.p.t. adversary can not extract any information about the witnesses other than the relation given the argument and the confidential transactions. Formally, it is captured by giving the adversary challenging inputs from a randomly chosen distribution of (1) a real distribution and (2) a simulated distribution and asking the adversary to distinguish which distribution it is. The simulated distribution depicts the real distribution with valid arguments. If the adversary cannot identify the distribution correctly with a more than $1/2 + \epsilon(\lambda)$ probability, then we conclude that our protocol is zero-knowledge. Here, $\mathcal{S}(pp_{\lambda,L}, \mathcal{T}, \mathcal{L})$ generates the simulated distribution given a trapdoor $\mathcal{T}$ of $pp_{\lambda,L}$.

Definition 8 (Zero-Knowledge Argument). *Let, $pp_{\lambda,L} \leftarrow \texttt{CIP.Set}()$ and $\mathcal{T}$ be a trapdoor of $pp_{\lambda,L}$. $D_0 = D(\mathcal{S}(pp_{\lambda,L}, \mathcal{T}, \mathcal{L}))$ is a simulation distribution generated by a simulator $\mathcal{S}(pp_{\lambda,L}, \mathcal{T}, \mathcal{L})$, and $D_1 = D(\texttt{CIP}(pp_{\lambda,L}, \mathcal{L}))$ is the real distribution of any relation in $\mathcal{L}$ and its confidential integers. $\texttt{CIP}$ provides zero-knowledge arguments for any p.p.t. adversary $\mathcal{A}$ if $\texttt{Adv}^{\texttt{ZK}}_{\texttt{CIP},\mathcal{A}}$ is,*

$$2\left|\frac{1}{2} - Pr\left[b \overset{?}{=} b' \;\middle|\; b \xleftarrow{\$} [0,1]; ([c_i]^*_{i=0}, \pi) \xleftarrow{\$} D_b; b' \leftarrow \mathcal{A}(pp_\lambda, [c_i]^*_{i=0}, \pi)\right]\right| \leq \epsilon(\lambda).$$

Intra-Privacy. Informally, intra-privacy is the ability to maintain secrets during the proving phase. For example, during the "multi-party equal" statements, none of the provers learns others' secret keys. Therefore, intra-privacy is the insiders' zero-knowledge property. Recall that our CIP's proving phase is an interactive multi-party protocol where the participants interact with each other on several occasions. Hence, we define intra-privacy to be zero-knowledge with multiple transcripts (or for multiple interactions). We consider a system with *only one honest participant* and $\mathcal{A}$ acts as the other participants since the adversary $\mathcal{A}$ is an insider. Also, for all transcripts, the honest participant chooses its transcripts $\theta_{t,0}$ first and gives them to $\mathcal{A}$ allowing to choose its transcripts $[\theta_{t,j}]^*_{j=1}$ according to $\theta_{t,0}$. Here, the adversary wins the game if it identifies simulated transcripts over properly generated transcripts with more than $1/2 + \epsilon(\lambda)$ probability. We formally define intra-privacy below.

Definition 9 (Intra-Privacy). *Let the honest participant's simulated input distribution be $\mathcal{D}_{0,t} = \mathcal{D}(\mathcal{S}(pp_{\lambda,L}, \mathcal{T}, \mathcal{L}; \pi_{t-1}))$ and real input distribution be $\mathcal{D}_{1,t} = \mathcal{D}(\texttt{CIP}(pp_{\lambda,L}, \mathcal{L}; \pi_{t-1}))$ for transcript t when $\mathcal{S}$ has access to a trapdoor $\mathcal{T}$ of $pp_{\lambda,L}$. $\texttt{CIP}$ provides intra-privacy for any p.p.t. adversary $\mathcal{A}$ if $\texttt{Adv}^{\texttt{IP}}_{\texttt{CIP},\mathcal{A}}$ is,*

$$2\left|\frac{1}{2} - Pr\left[b \overset{?}{=} b' \;\middle|\; \begin{array}{r} b \xleftarrow{\$} [0,1]; \forall t : \left\{\begin{array}{r} \theta_{t,0} \xleftarrow{\$} \mathcal{D}_{b,t}; [\theta_{t,j}]^*_{j=1} \leftarrow \mathcal{A}(pp_{\lambda,L}, \pi_{t-1}, \theta_{t,0}) \\ \pi_t := \texttt{CIP.combine}_t(r, \pi_{t-1}, \theta_{t,0}, [\theta_{t,j}]^*_{j=1}) \end{array}\right\} \\ b' \leftarrow \mathcal{A}(pp_\lambda, \pi_0, \pi_{\texttt{final}}) \end{array}\right]\right| \leq \epsilon(\lambda).$$

3.2 Hybrid Model of CIP

Building universally composable protocols from scratch is known to be hard due to UC's strict security notions, and complex protocols like ours commonly

Algorithm 1 Ideal Function $\mathcal{F}_{\texttt{CIP}}$ (Part I)

Records: This ideal function keeps a table $\boldsymbol{C}$ for records.

Extracting Setup: We consider an instance of a generic $\texttt{CIP}$ where the setup is predefined including the security parameter, λ and the public parameters, $pp_{\lambda,L}$. Hence, the setup is read-only, participants can get public parameters by sending a request to $\mathcal{F}_{\texttt{CIP}}$. Upon receiving $\texttt{req}_{\texttt{set}}(\texttt{set}, \texttt{uid})$ from a valid $\texttt{uid}$, $\mathcal{F}_{\texttt{CIP}}$ sends $\texttt{log}_{\texttt{set}}(\texttt{set}, \texttt{uid})$ to $\mathcal{S}_{\mathcal{A}}$, and gets $\texttt{ack}_{\texttt{set}}(\texttt{set}, \texttt{uid}, pp_{\lambda,L})$ from $\mathcal{S}_{\mathcal{A}}$. Finally, $\mathcal{F}_{\texttt{CIP}}$ sends $\texttt{res}_{\texttt{set}}(\texttt{set}, \texttt{uid}, pp_{\lambda,L})$ to $\mathcal{D}(\texttt{uid})$.

Sending Data: Since CIP is a multi-party system which has multiple provers and multiple verifiers (or both at once), we define a method to send data, e.g., confidential integers, openings, or arguments from a dummy participant $\mathcal{D}(\texttt{uid})$ to $\mathcal{D}(\texttt{uid}')$. The simulated adversary can delay, stop, eavesdrop or alter the communication according to the real adversary.

1) Sending Confidential Integers or Arguments: When $\mathcal{F}_{\texttt{CIP}}$ gets $\texttt{req}_{\texttt{c/psend}}($ $\texttt{c/psend}$, $\texttt{uid}, \texttt{uid}', c/\pi)$ for valid $(\texttt{uid}, \texttt{uid}')$, it forwards $\texttt{log}_{\texttt{c/psend}} = \texttt{req}_{\texttt{c/psend}}$ to $\mathcal{S}_{\mathcal{A}}$, and gets $\texttt{ack}_{\texttt{c/psend}}(\texttt{c/psend}, \texttt{uid}, \texttt{uid}'', c'/\pi')$ from $\mathcal{S}_{\mathcal{A}}$. If $\texttt{uid}''$ is invalid or $\perp$, halts. Otherwise, $\mathcal{F}_{\texttt{CIP}}$ sends $\texttt{res}_{\texttt{c/psend}}(\texttt{c/psend}, \texttt{uid}, \texttt{uid}'', c'/\pi')$ to $\mathcal{D}(\texttt{uid}'')$.

2) Sending Openings: Dummy parties send the commitment when they want to send the opening. Upon receiving $\texttt{req}_{\texttt{op\_send}}(\texttt{op\_send}, \texttt{uid}, \texttt{uid}', c)$ for valid $(\texttt{uid}, \texttt{uid}')$, $\mathcal{F}_{\texttt{CIP}}$ sends $\texttt{log}_{\texttt{op\_send}}$=$\texttt{req}_{\texttt{op\_send}}$ to $\mathcal{S}_{\mathcal{A}}$, and gets $\texttt{ack}_{\texttt{op\_send}}(\texttt{op\_send}, \texttt{uid}, \texttt{uid}'', c')$. If $c' \neq \perp$ and $\texttt{uid}''$ is valid, sends $\texttt{res}_{\texttt{op\_send}}(\texttt{op\_send}, \texttt{uid}, \texttt{uid}'', c')$ to $\mathcal{D}(\texttt{uid}'')$.

Confidential Integer Generation: We can use this to commit an integer. When $\mathcal{F}_{\texttt{CIP}}$ receives $\texttt{req}_{\texttt{cint}}(\texttt{cint}, \texttt{uid}, v)$ from a valid $\texttt{uid}$, sends $\texttt{log}_{\texttt{cint}}(\texttt{cint}, \texttt{uid}, v)$ to $\mathcal{S}_{\mathcal{A}}$, and receives $\texttt{ack}_{\texttt{cint}}(\texttt{cint}, \texttt{uid}, v, k, c)$ from $\mathcal{S}_{\mathcal{A}}$. $\mathcal{F}_{\texttt{CIP}}$ identifies whether there are $[(*, c_i, v_i, k_i) \in \boldsymbol{C}]_{i=0^*}$ such that $\sum_{i=0}^{*} \pm c_i = c$, i.e., c has an additive relation to the previously generated confidential integers.

```
  (Hamiltonicity) if $(*, *, *, k) \in \boldsymbol{C}$: sends $\texttt{res}_{\texttt{cint}}(\texttt{cint}, \texttt{uid}, v, \perp)$ and halt.
  (Binding) if $(c_0 \pm_c .. \pm_c c_*) = c$ s.t. $(\sum_{i=0}^{*} \pm v_i \neq v \vee (k_0 \pm_k .. \pm_k k_*) \neq k)$:
      sends $\texttt{res}_{\texttt{cint}}(\texttt{cint}, \texttt{uid}, v, \perp)$ and halt.
  else sends $\texttt{res}_{\texttt{cint}}(\texttt{cint}, \texttt{uid}, v, c)$ and adds $(\texttt{uid}, v, k, c)$ to $\boldsymbol{C}$ if $c \neq \perp$.
If $b = 0$, $\mathcal{F}_{\texttt{CIP}}$ sends $\texttt{res}_{\texttt{cint}}(\texttt{cint}, \texttt{uid}, v, c)$.
```

Confidential Integer Opening: We can open a confidential integer with the opening. After receiving a request $\texttt{req}_{\texttt{open}}(\texttt{open}, \texttt{uid}, c)$, $\mathcal{F}_{\texttt{CIP}}$ sends $\texttt{log}_{\texttt{open}}(\texttt{open}, \texttt{uid}, c)$ to $\mathcal{S}_{\mathcal{A}}$, and receives $\texttt{ack}_{\texttt{open}}(\texttt{open}, \texttt{uid}, c, v, k, b)$ from $\mathcal{S}_{\mathcal{A}}$. Checks if c has a relation with previous confidential integers such that $\sum_{i=0}^{*} \pm c_i = c$ for some $[(*, c_i, v_i, k_i) \in \boldsymbol{C}]_{i=0}^{*}$.

```
  (Completeness) if $(c_0 \pm_c .. \pm_c c_*) = c$ s.t. $(\sum_{i=0}^{*} \pm v_i = v \vee (k_0 \pm_k .. \pm_k k_*) = k)$:
      sends $\texttt{res}_{\texttt{open}}(\texttt{open}, \texttt{uid}, c, 1)$ and halts.
  (Binding) if $b = 1 \wedge (c_0 \pm_c .. \pm_c c_*) = c$ s.t. $(\sum_{i=0}^{*} \pm v_i \neq v \vee (k_0 \pm_k .. \pm_k k_*) \neq k)$:
      sends $\texttt{res}_{\texttt{open}}(\texttt{open}, \texttt{uid}, c, 0)$ and halts.
  else sends $\texttt{res}_{\texttt{open}}(\texttt{open}, \texttt{uid}, c, b)$ and adds $(\texttt{uid}, v, k, c)$ to $\boldsymbol{C}$ if $b = 1$.
```

use “helping functionalities”. Zero-knowledge and simulation extractability are popular examples of helping functionalities since commitments can not achieve zero-knowledge or simulation extractability in the plain UC model. This hybrid model was first explained in [17] showing that a UC commitment can be built on Blum’s Graph Hamiltonicity [6]. A Hamiltonian path, a.k.a., a traceable

Algorithm 2 Ideal Functionality of $\mathcal{F}_{\texttt{CIP}}$: $\mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}$ (Part II)

Records: This ideal function keeps tables $\boldsymbol{\Theta}$ and $\boldsymbol{\Pi}$ for records.

Independent Public Transcript Generation: Since some of these proving functions are multi-party computations, we use $\texttt{sid}$ to identify each of these multi-party computation separately. A dummy participant $\mathcal{D}(\texttt{uid})$ can generate transcripts by sending $\texttt{req}_{\texttt{prove}}(\texttt{prove}_t, \texttt{uid}, \texttt{sid}, t, r, \pi_{t-1})$ for the (t)th transcript of session $\texttt{sid}$. After receiving the request, $\mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}$ sends $\texttt{log}_{\texttt{prove}}(\texttt{prove}_t, \texttt{uid}, \texttt{sid}, t, r, \pi_{t-1})$ to $\mathcal{S}_{\mathcal{A}}$, and gets $\texttt{ack}_{\texttt{prove}}(\texttt{ack}_t, \texttt{uid}, \texttt{sid}, t,\ r, \pi_{t-1}, \theta_{t,\texttt{uid}})$.

(Hamiltonicity) **if** $(*, *, \theta_{t,\texttt{uid}}, *) \in \boldsymbol{\Theta}$: sends $\texttt{res}_{\texttt{prove}}(\texttt{prove}_t, \texttt{uid}, \texttt{sid}, t, r, \pi_{t-1}, \perp)$. Else sends $\texttt{res}_{\texttt{prove}}(\texttt{prove}_t, \texttt{uid}, \texttt{sid}, t, r, \pi_{t-1}, \theta_{t,\texttt{uid}})$. Also, $\mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}$ adds $(\texttt{sid}, t, \theta_{t,\texttt{uid}}, \pi_{t-1})$ to $\boldsymbol{\Theta}$ if $\theta_{t,\texttt{uid}}$ is not $\perp$.

(Partial-)Argument Generation: Once a participant collects all of public transcripts for the session, that participant can combine them into a argument or partial argument. Assume $\mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}$ receives $\texttt{req}_{\texttt{combine}}(\texttt{combine}_t, \texttt{uid}, \texttt{sid}, t, r, \pi_{t-1}, \boldsymbol{\theta}_t)$ from a valid $\texttt{uid}$. Then $\mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}$ sends $\texttt{log}_{\texttt{combine}}(\texttt{combine}_t, \texttt{uid}, \texttt{sid}, t, r, \pi_{t-1}, \boldsymbol{\theta}_t)$ to $\mathcal{S}_{\mathcal{A}}$ and gets $\texttt{ack}_{\texttt{combine}}(\texttt{combine}_t, \texttt{uid}, \texttt{sid}, t, r, \pi_{t-1}, \boldsymbol{\theta}_t, \pi_t)$.

```
(Consistency) if $(\texttt{sid}, t, \pi_{t-1}, \boldsymbol{\theta}_t, \pi'_t \neq \pi_t) \in \boldsymbol{\Pi}$:
    sends $\texttt{res}_{\texttt{combine}}(\texttt{combine}_t, \texttt{uid}, \texttt{sid}, t, r, \pi_{t-1}, \boldsymbol{\theta}_t, \pi'_t)$.
else: sends $\texttt{res}_{\texttt{combine}} := \texttt{ack}_{\texttt{combine}}$ and adds $(\texttt{sid}, t, \pi_{t-1}, \boldsymbol{\theta}_t, \pi_t)$ to $\boldsymbol{\Pi}$ if $\pi_t \neq \perp$.
```

Final Relation Argument Verification: Once a final argument for a relation $r \in \mathcal{L}$ is given with its input confidential integers, we should be able to verify whether the committed values of confidential integers hold r or not without actually seeing them. Note that all verification functions are single-party computations; hence, we do not need session identities. Assume $\mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}$ receives $\texttt{req}_{\texttt{ver}}(\texttt{ver}, \texttt{uid}, r, [c_i]^*_{i=0}, \pi)$. Then $\mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}$ sends $\texttt{log}_{\texttt{ver}}(\texttt{ver}, \texttt{uid}, r, [c_i]^*_{i=0}, \pi)$ to $\mathcal{S}_{\mathcal{A}}$ and gets $\texttt{ack}_{\texttt{ver}}(\texttt{ver}, \texttt{uid}, r, [c_i]^*_{i=0}, \pi, b)$.

```
(Completeness) if $(\texttt{sid}, 0, *, [c'_i]^*_{i=0}, *, *) \wedge (\texttt{sid}, \texttt{final}, *, *, \pi) \in \boldsymbol{\Pi}$ for some $\texttt{sid}$
or $(-, \texttt{final}, [c_i]^*_{i=0}, -, \pi) \in \boldsymbol{\Pi}$: sends $\texttt{res}_{\texttt{ver}}(\texttt{ver}, \texttt{uid}, r, [c_i]^*_{i=0}, \pi, 1)$
else: sends $\texttt{log}_{\texttt{ver}}(\texttt{ver}, \texttt{uid}, r, [c_i]^*_{i=0}, \pi, b)$ and adds $(-, \texttt{final}, [c_i]^*_{i=0}, -, \pi)$ if $b = 1$.
```

path of a graph, visits each vertex exactly once and is used to create indistinguishable distributions for cheating provers, i.e., provers who want to break the zero-knowledge argument. However, this method does not work when the commitment is malleable, like additively homomorphic commitments. There are a few alternatives; (1) a "catalyst" [32] which acts as a primitive function and can be used without any precautions between the systems, or (2) setting trusted setups like trapdoors. We use Blum's Graph Hamiltonicity for relations' transcripts and secret keys, not for commitments.

Our ideal system has three components; (1) an ideal function $\mathcal{F}_{\texttt{CIP}}$ in Algorithm 1 that is binding with Hamiltonian keys (or one-time keys) for confidential integers, (2) a helping ideal functionality $\mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}$ in Algorithm 2 to ensure Hamiltonicity of transcripts, and (3) a simulated adversary $\mathcal{S}_{\mathcal{A}}$ who creates an opportunity for $\mathcal{E}$ to identify the system if the real $\texttt{CIP}$ is not a zero-knowledge argument with intra-privacy and knowledge soundness by exposing $\mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}$.

Algorithm 1 explains the generic properties of $\mathcal{F}_{\texttt{CIP}}$ like how dummy parties extract the setup and share data among each other in an environment that can be intervened by an adversary. In Algorithm 1, $\mathcal{F}_{\texttt{CIP}}$ answers $\texttt{req}_{\texttt{cint}}$ and $\texttt{req}_{\texttt{open}}$. Dummy participants can generate confidential integers for some integer by sending $\texttt{req}_{\texttt{cint}}$. The ideal function replies with a confidential integer without the secret key. During the generation, the ideal function makes sure that honestly generated confidential integers are valid (the completeness), and secret keys are one-time. The most important property is the binding property. The ideal function treats the first confidential integer and its opening as valid, and any other opening(s) for the same commitment is considered invalid. Note that the binding property is not limited to identifying two openings for the same commitment but any additive commitment created from previously generated commitments. Therefore, the ideal function checks if the given confidential integer can be recreated from the previously generated commitments or not. If it can be recreated, but the recreated commitment's opening is different, then the ideal function rejects the confidential integer.

We describe $\mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}$ in Algorithm 2. Recall that our proving algorithm can be multi-party and have multiple steps, e.g., composed relations. Therefore, $\mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}$ allows $\texttt{req}_{\texttt{prove}}$ requests to be multi-party by defining a unique session identity $\texttt{sid}$. Explicitly, $\mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}$ shows that if a relation's argument is created through the proving function, it is always valid, and the transcripts are Hamiltonian. Before explaining other security properties, we need to understand how $\mathcal{S}_{\mathcal{A}}$ works for the composed system $\{\mathcal{F}_{\texttt{CIP}}, \mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}\}$.

Simulated Adversary $\mathcal{S}_{\mathcal{A}}$. In the real system, the adversaries can act as normal participants yet try to exploit the system's weakness. $\mathcal{S}_{\mathcal{A}}$'s objective is to simulate those malicious participants in the ideal world. As we have shown in Figure 2, $\mathcal{S}_{\mathcal{A}}$ runs a copy of $\mathcal{A}$'s iTM inside the ideal system and acts accordingly when $\mathcal{E}$ interacts with the ideal system. We formally define each action of $\mathcal{S}_{\mathcal{A}}$ in Figure 3, i.e., general actions and acknowledgements. The main objective of $\mathcal{S}_{\mathcal{A}}$ is to show whether $\texttt{CIP}$ is zero-knowledge, knowledge sound, and intra-private or not. We start with the zero-knowledge argument (Definition 8). $\mathcal{S}_{\mathcal{A}}$ simulates the same scenario by sending randomly chosen simulated arguments and properly generated arguments when the prover is corrupted. Similarly, $\mathcal{S}_{\mathcal{A}}$ sends randomly chosen simulated transcripts and properly generated transcripts to corrupted provers imitating the scenario in intra-privacy (Definition 9). $\mathcal{S}_{\mathcal{A}}$ creates the behavior of simulation extractability (Definition 7), which says that cheating provers can not create valid arguments for invalid relations by asking the extractor $\mathcal{K}$ to check validity with its trapdoor if the prover is corrupt, and the argument is not simulated. However, $\mathcal{S}_{\mathcal{A}}$ acts properly for honest participants (uses $\texttt{CIP}$'s functions), forcing the protocol to be correct.

In the next section, we will explain CIP-DLP's and CIP-SIS's concrete protocols that realize the defined ideal function.

Algorithm 3 The Simulated Adversary

1) General Actions. $\mathcal{S}_{\mathcal{A}}$ keeps the public parameter set $pp_{\lambda,L}$, a trapdoor $\mathcal{T}$ for $pp_{\lambda,L}$, and two tables; $\boldsymbol{K}$ and $\boldsymbol{\Theta}$. If $\mathcal{A}$ corrupts a real participant of $\mathcal{P}(\mathtt{CIP},\mathtt{uid})$, then $\mathcal{S}_{\mathcal{A}}$ corrupts (gives the access to all input and output tapes) the dummy participant $\mathcal{D}(\mathtt{uid})$ and remembers that $\mathcal{D}(\mathtt{uid})$'$\mathtt{state}$ is $\mathtt{corrupt}$. When $\mathcal{S}_{\mathcal{A}}$ receives $\mathtt{log}_{\mathtt{set}}(\mathtt{set},\mathtt{uid})$ from $\mathcal{F}_{\mathtt{CIP}}$, it replies $\mathtt{ack}_{\mathtt{set}}(\mathtt{set},\mathtt{uid},pp_{\lambda,L})$. If $\mathcal{S}_{\mathcal{A}}$ gets $\mathtt{log}_{\mathtt{c/psend}}(\mathtt{c/psend},\mathtt{uid},\mathtt{uid}',c/\pi)$ it checks if $\mathcal{A}$ delays/eavesdrops/alters the communication, $\mathcal{S}_{\mathcal{A}}$ sends delayed/open/altered $\mathtt{ack}$ to $\mathcal{F}_{\mathtt{CIP}}$, e.g., if $\mathcal{A}$ changes c to c', the acknowledgement is $\mathtt{ack}_{\mathtt{c/psend}}(\mathtt{c/psend},\mathtt{uid},\mathtt{uid}',c')$. For an opening sharing log; $\mathtt{log}_{\mathtt{op\_send}}(\mathtt{op\_send},\mathtt{uid},\mathtt{uid}',c)$, $\mathcal{S}_{\mathcal{A}}$ checks if c is stored in $\boldsymbol{K}$ with its opening (v,k) such that $(b,c,v,k,\mathtt{uid})\in\boldsymbol{K}$. $\mathcal{S}_{\mathcal{A}}$ works as follows.

- **if** $\mathcal{A}$ does intervene: sets $\mathit{out}=(\mathtt{uid}'',c')$ according to $\mathcal{A}$. If $(b,c',v',k',*)\in\boldsymbol{K}$ then adds $(b,c',v',k',\mathtt{uid}'')$ to $\boldsymbol{K}$. Else, adds $(0,c',v',k',\mathtt{uid}'')$ to $\boldsymbol{K}$.
- **if** $(1,c,v,k,\mathtt{uid})\in\boldsymbol{K}$: sets $\mathit{out}=(\mathtt{uid}',c)$ and adds $(1,c,v,k,\mathtt{uid}')$
- **else** $(1,c,v,k,\mathtt{uid})\notin\boldsymbol{K}$: sets $\mathit{out}=(\mathtt{uid}',\perp)$.

Then $\mathcal{S}_{\mathcal{A}}$ sends $\mathtt{ack}_{\mathtt{op\_send}}(\mathtt{op\_send},\mathtt{uid},\mathit{out})$ to $\mathcal{F}_{\mathtt{CIP}}$. When $\mathcal{A}$ changes any openings saved in real participants, $\mathcal{S}_{\mathcal{A}}$ changes $\boldsymbol{K}$ accordingly, e.g., if (c,v,k) is deleted from $\mathcal{P}(\mathtt{CIP},\mathtt{uid})$, $\mathcal{S}_{\mathcal{A}}$ removes $(*,c,v,k,\mathtt{uid})$ from $\boldsymbol{K}$. Similarly, if (c,v,k) is changed into (c',v',k') such that $(1,c',v',k',\mathtt{uid}')\notin\boldsymbol{K}$ for any corrupt $\mathtt{uid}'$ then $(0,c',v',k',\mathtt{uid}')$ will be added and $(*,c,v,k,\mathtt{uid})$ will be deleted from $\boldsymbol{K}$.

2) Actions for Confidential Integers. For $\mathtt{log}_{\mathtt{cint}}(\mathtt{cint},\mathtt{uid},v)$, $\mathcal{S}_{\mathcal{A}}$ sets $b=1$ when $\mathtt{uid}$ is honest or else picks $b\xleftarrow{\$}\{0,1\}$. If $b=1$, computes $\mathtt{CIP.cint}(v)\rightarrow(k,c)$ or else gets $(k,c)\xleftarrow{\$}\mathcal{S}(pp_{\lambda,L},\mathcal{T},\mathcal{L},v)$. Sends $\mathtt{ack}_{\mathtt{cint}}(\mathtt{cint},\mathtt{uid},v,k,c)$ to $\mathcal{F}_{\mathtt{CIP}}$, and adds $(1,c,v,k,\mathtt{uid})$ to $\boldsymbol{K}$. For $\mathtt{log}_{\mathtt{open}}(\mathtt{open},\mathtt{uid},c)$, $\mathcal{S}_{\mathcal{A}}$ check if $(*,c,v,k,\mathtt{uid})\in\boldsymbol{K}$ for some v and k then computes $\mathtt{CIP.open}(c,v,k)=b$ and sends $\mathtt{log}_{\mathtt{open}}(\ \mathtt{open},\mathtt{uid},c,v,k,b)$ to $\mathcal{F}_{\mathtt{CIP}}$. Otherwise, $\mathcal{S}_{\mathcal{A}}$ sends $\mathtt{log}_{\mathtt{open}}(\ \mathtt{open},\mathtt{uid},c,v,k,0)$.

3) Actions for Relations. The simulated adversary treats corrupted provers differently. When $\mathcal{S}_{\mathcal{A}}$ gets $\mathtt{log}_{\mathtt{prove}}(\mathtt{prove}_t,\mathtt{uid},\mathtt{sid},t,r,\pi_{t-1})$ from $\mathcal{F}^{\mathcal{L}}_{\mathtt{ZK,KS,IP}}$, if $t=1$ and $\mathtt{sid}$ does not have corrupted participant(s), sets $b=1$. Else, picks $b\xleftarrow{\$}\{0,1\}$ if $t=1$.

- **if** $(b',\mathtt{sid},\mathtt{uid},t,*,*,*)\notin\boldsymbol{\Theta}\wedge t\neq 0$: sets $\theta_{t-1,}=\perp$ and $b:=b'$
- **if** $t=1\wedge b=1$: tries to find the opening for the dummy party's confidential integer $c_{\mathtt{uid}}\in\pi_0$ such that $(1,c_{\mathtt{uid}},v,k,\ \mathtt{uid})\in\boldsymbol{K}$. If $\mathcal{S}_{\mathcal{A}}$ can't find the opening, sets $\theta_t\ =\perp$. If $\mathcal{S}_{\mathcal{A}}$ finds the opening, sets $\boldsymbol{k}_0=(v,k)$, computes $(\boldsymbol{k}_1,\theta_1)=\mathtt{CIP.prove}_t(r;\ \pi_0,\boldsymbol{k}_0)$ and adds $(1,\mathtt{sid},\mathtt{uid},0,\ \pi_0,\theta_1,\boldsymbol{k}_1)$ to $\boldsymbol{\Theta}$.
- **if** $t=1\wedge b=0$: sets $\theta_1\xleftarrow{\$}\mathcal{S}(pp_{\lambda,L},\mathcal{T},\mathcal{L},\pi_0)$ and adds $(0,\mathtt{sid},\mathtt{uid},0,\ \pi_0,\theta_1,-)$.
- **if** $b=0$: sets $\theta_t\xleftarrow{\$}\mathcal{S}(pp_{\lambda,L},\mathcal{T},\mathcal{L},\pi_{t-1})$ and adds $(0,\mathtt{sid},\mathtt{uid},t,\ \pi_{t-1},\theta_t,-)$ to $\boldsymbol{\Theta}$.
- **else**: tries to find secret keys of the previous transcript such that $(1,\mathtt{sid},\mathtt{uid},t-1,\ \pi_{t-2},\theta_{t-1},\boldsymbol{k}_{t-1})$. If $\mathcal{S}_{\mathcal{A}}$ cannot find the opening, sets $\theta_t=\perp$. Else, computes $(\boldsymbol{k}_t,\theta_t)=\mathtt{CIP.prove}_t(r;\ \pi_{t-1},\boldsymbol{k}_{t-1})$ and adds $(1,\mathtt{sid},\mathtt{uid},t,\ \pi_{t-1},\theta_t,\boldsymbol{k}_t)$ to $\boldsymbol{\Theta}$.

Finally, $\mathcal{S}_{\mathcal{A}}$ sends $\mathtt{ack}_{\mathtt{prove}}(\mathtt{prove}_t,\mathtt{uid},\mathtt{sid},t,r,\pi_{t-1},\theta_t)$ to $\mathcal{F}^{\mathcal{L}}_{\mathtt{ZK,KS,IP}}$.

The simulated adversary combines transcripts of both distributions exactly the same way, i.e., $\mathcal{S}_{\mathcal{A}}$ replies with $\mathtt{ack}_{\mathtt{combine}}(\mathtt{combine}_t,\mathtt{uid},\mathtt{sid},t,r,\pi_{t-1},\boldsymbol{\theta}_t,\pi_t{=}\mathtt{CIP.prove}_t(r;\ \pi_{t-1},\boldsymbol{\theta}_t))$ for $\mathtt{log}_{\mathtt{combine}}(\mathtt{combine}_t,\mathtt{uid},\mathtt{sid},t,r,\pi_{t-1},\boldsymbol{\theta}_t)$. $\mathcal{S}_{\mathcal{A}}$ works as follows for $\mathtt{log}_{\mathtt{ver}}(\mathtt{ver}_t,\mathtt{uid},r,\pi_0,\pi)$. When there are $(0,\mathtt{sid},*,t,\pi_{t-1},*)\in\boldsymbol{\Theta}$ and $(0,\mathtt{sid},*,\mathtt{final},*,\pi)\in\boldsymbol{\Theta}$, sets $b=1$. $\mathcal{S}_{\mathcal{A}}$ sets $b=r(\mathcal{K}(pp_{\lambda,L},\mathcal{T},r;\pi_0,\pi))$ if $\mathcal{D}(\mathtt{uid})$ is $\mathtt{corrupt}$. Otherwise, $b=\mathtt{CIP.ver}(r;\pi_0,\pi)$. Finally, $\mathcal{S}_{\mathcal{A}}$ sends $\mathtt{ack}_{\mathtt{ver}}(\mathtt{ver}_t,\mathtt{uid},r,\pi_0,\pi,b)$.

4 Confidential Integer Processing Protocols

Many confidential blockchains are either based on the Discrete-Logarithmic Problem (DLP) or Shortest Integer Solution Problem (SIS). Therefore, we propose two implementations; CIP-DLP from DLP and CIP-SIS from Approx-SIS.

We compute public parameters for integers in $(-2^L, 2^L)$. However, the range proofs can be created for any customized range up to $(2L+5)$-bits since range proofs are used in "not equal" and "signed division". All the relations are modular operations, e.g., the addition of $v_0 = v_1 + v_1$ means $v_0 = v_1 + v_2 \pmod{q}$ when $q > 2^{2L+5}$ is the modulus of the system. Hence, the integers can exceed the range $(-2^L, 2^L)$ during the operations. However, the users can restrict the range to the desired range by creating range proofs, e.g., a composed relation of an addition and a range proof for the output. We leave this option to the users.

We use `hash()` to denote a hash function family with co-domains of $\mathbb{Z}_q^+$ for CIP-DLP and $\mathcal{C}_\beta^N$ for CIP-SIS, which will be clearly defined at each step.

```
CIP_DLP.setup():
▷ $\mathbb{G} = \langle g \rangle = \langle h \rangle$ is a group of prime order $q \geq \{0,1\}^\lambda$, and the discrete logarithms of $g$ and $h$ relative to each other are unknown — or $g$ and $h$ are "nothing-up-my-sleeve" (NUMS) group generators. $\boldsymbol{g}$ and $\boldsymbol{h}$ are generator vectors of $\mathbb{G}$ where $|\boldsymbol{g}| = |\boldsymbol{h}| = 2L+5$.
return $pp = (q > 2^{2L+5}, g, h, \boldsymbol{g}, \boldsymbol{h}, L)$.
```

```
CIP_SIS.setup():
Parameters $n, m, q, \gamma, N, \gamma', \mathcal{C}_\beta^N, \alpha, \alpha_1, \tau, \tau_1, \tau_2, \tau_3, p_0, p_1, \mathcal{X}$ are chosen such that
    prime $q \equiv 1 \pmod{2N}$
    ▷ for negligible soundness error
    $\log_2 \binom{N}{\beta} + \beta \geq 2\lambda$,
    ▷ for rejection sampling in proofs
    $2 \ll \alpha$, $2^L \ll \alpha_1$, $\beta\tau + \beta^2\tau_1 \ll \tau_2 \leq \gamma$,
    $0 < 4\beta^2\tau + \beta\tau_1 \ll \tau_2 \leq \gamma$, $0 < \beta\tau \ll \tau_3$
    ▷ for computational hiding
    $mN \log_2(2\tau) \geq 6\lambda$,
    $2^{2(L+1)}\alpha \ll \gamma$, $N\alpha_1^2 + 2N\beta\alpha_2 < \gamma$,
    ▷ to keep error within $\gamma$ and facilitate $2^{32}$ transactions
    $p_0 \cdot 2^{32} \leq \gamma' \ \wedge \ \beta * N * p_1 \leq \gamma'$
    ▷ to generate hints within a reasonable time (heuristically)
    $\mathcal{X} \ll n * N$
$\vec{\boldsymbol{K}} \xleftarrow{\$} \mathbb{X}_q^{n \times m}$, $\vec{\boldsymbol{V}} \xleftarrow{\$} \mathbb{X}_q^n$, $\vec{\boldsymbol{R}} \xleftarrow{\$} \mathbb{X}_q^n$ such that $\vec{\boldsymbol{H}} = [\vec{\boldsymbol{K}}, \vec{\boldsymbol{V}}, \vec{\boldsymbol{R}}]$
▷ challenge space
$\mathcal{C}_\beta^N = [x \in \mathbb{X}_q \text{ s.t. } \|x\| = 1, \|x\|_1 = \beta]$
return $pp_\lambda = (L, n, m, q, N, \gamma, \gamma', \mathcal{C}_\beta^N, \alpha, \tau, \tau_1, \tau_2, \tau_3, p_0, p_1, \vec{\boldsymbol{K}}, \vec{\boldsymbol{V}}, \vec{\boldsymbol{R}})$
```

Confidential Integers. We define two confidential integers from Pedersen commitments and multiplications of Approx-SIS hard matrices when the range is limited to $(-2^L, 2^L)$ (Appendix B explains how to prove that a confidential integer holds the same value as a binary lattice confidential coin).

```
CIP_DLP.cint($v$) :
if $v \notin (-2^L, 2^L)$ : return $\perp$
$k \xleftarrow{\$} \mathbb{Z}_q^+$; return $g^k h^v \pmod{q} \in \mathbb{G}$
```

```
CIP_DLP.open($c, v, k$) :
return $c \overset{?}{=} g^k h^v \pmod{q} \wedge v \overset{?}{\in} (-2^L, 2^L)$
```

```
CIP_SIS.cint($v$) :
if $v \notin (-2^L, 2^L)$ : return $\perp$
$\vec{\boldsymbol{k}} \xleftarrow{\$} [-\tau, \tau]^{m \times N}$; return $\vec{\boldsymbol{c}} = \mathtt{up}_{p_0}(\mathtt{highbits}_{p_0}(\vec{\boldsymbol{V}}[v, 0, .., 0] + \vec{\boldsymbol{K}}\vec{\boldsymbol{k}})) \in \mathbb{X}_q^n$
```

```
CIP_SIS.open($\vec{\boldsymbol{c}}, v, \vec{\boldsymbol{k}}$) : return $v \overset{?}{\in} (-2^L, 2^L)$
    $\wedge \|\vec{\boldsymbol{k}}\| \leq \tau \wedge \vec{\boldsymbol{c}} \overset{?}{=} \mathtt{up}_{p_0}(\mathtt{highbits}_{p_0}(\vec{\boldsymbol{V}}[v, 0, .., 0] + \vec{\boldsymbol{K}}\vec{\boldsymbol{k}}))$
```

As we show in confidential integers, the range of a confidential integers can not be verified unless it is opened. However, we do not want to open confidential integers during public verification. Therefore, we use range proofs to verify the range of a confidential integer. Not only that, range proofs are heavily used in the following proofs; “not equal”, “greater/less than equal”, and “signed division”. Therefore, we configure range proofs for integers in $[0, 2^{2L+5})$, i.e., the custom range L' can be in $[0, 2L+5]$. For example, the recommended range $(-2^L, 2^L)$ of an confidential integer $c(v,k)$ can be verified through a range proof of $(c(v,k) + c(2^L, 0))$ when $L' = L+1$ and $c(2^L, 0)$ does not have a secret key.

We use a non-interactive version of Bullet-proof Range Proofs (BRP) [14] for $\texttt{CIP}_{\texttt{DLP}}$ (Appendix C), and define a **novel** range proof scheme for $\texttt{CIP}_{\texttt{SIS}}$.

```
▷ Range Proof Generation
$\underline{\texttt{CIP}_{\texttt{DLP}}.\texttt{prove}(\texttt{range}_{L'}; \pi_0{=}c, \boldsymbol{k}{=}\{v, k\}) :}$
return $\pi := \texttt{BRP.prv}(pp_{\lambda,L'}, c, v, k)$

▷ Range Proof Verification
$\underline{\texttt{CIP}_{\texttt{DLP}}.\texttt{ver}(\texttt{range}_{L'}; c, \pi) :}$
return $\texttt{BRP.ver}(pp_{\lambda,L'}, c, \pi)$

▷ Range Proof Generation
 1: $\underline{\texttt{CIP}_{\texttt{SIS}}.\texttt{prove}(\texttt{range}_{L'}; \pi_0{=}\vec{\boldsymbol{c}}, \boldsymbol{k}{=}\{v, \vec{\boldsymbol{k}}\}):}$
 2: $\vec{b} = \mathsf{bin}(v)$ s.t. $\sum_{i=0}^{L'-1} 2^i b_i = v$
 3: $\vec{\boldsymbol{k}}_1 \xleftarrow{\$} [-\tau_1, \tau_1]^{m\times N}$; $\vec{\boldsymbol{k}}_2 \xleftarrow{\$} [-\tau_2, \tau_2]^{m\times N}$
 4: ▷ Lazy Sampling
 5: $\vec{\boldsymbol{a}} \xleftarrow{\$} [[-(\alpha-(1-b_i)), (\alpha-(1-b_i))]^N]_{i=0}^{L'-1}]$
 6: $\vec{\boldsymbol{s}}_0 = \sum_{i=0}^{L'-1} \vec{\boldsymbol{a}}_i \mathsf{rot}(2b_i - 1, i) \in \mathbb{X}$
 7: $\vec{\boldsymbol{t}}_1 = \texttt{highbits}_{p_1}(\vec{\boldsymbol{K}}\vec{\boldsymbol{k}}_1 + \vec{\boldsymbol{R}}\vec{\boldsymbol{s}}_0) \in \mathbb{X}_q^n$
 8: ▷ Challenge 1
 9: $\vec{x}_1 = \mathsf{hash}(\vec{\boldsymbol{c}}, \vec{\boldsymbol{t}}_1) \in \mathcal{C}_\beta^N$
10: $\vec{s}_1 = \sum_{i=0}^{L'-1} 2^i \vec{\boldsymbol{a}}_i \in \mathbb{X}$
11: $\vec{s}_2 = \vec{x}_1 (\sum_{i=0}^{L'-1} (\vec{\boldsymbol{a}}_i \mathsf{rot}(1, i))^2) \in \mathbb{X}$
12: $\vec{\boldsymbol{t}}_2{=}\texttt{highbits}_{\gamma'}(\vec{\boldsymbol{K}}\vec{\boldsymbol{k}}_2{+}\vec{\boldsymbol{V}}\vec{s}_1{+}\vec{\boldsymbol{R}}\vec{s}_2){\in}\mathbb{X}_q^n$
13: ▷ Challenge 2
14: $\vec{x}_2{=}\mathsf{hash}(\vec{x}_1, \vec{\boldsymbol{t}}_2) \in \mathcal{C}_\beta^N$
15: $[\vec{\boldsymbol{z}}_i = \vec{x}_2 \mathsf{rot}(b_i, 0) + \vec{\boldsymbol{a}}_i]_{i=0}^{L'-1} \in \mathbb{X}^{L'}$
16: $\vec{\boldsymbol{s}} = \vec{x}_2(\vec{\boldsymbol{k}} + \vec{x}_1\vec{\boldsymbol{k}}_1) + \vec{\boldsymbol{k}}_2 \in \mathbb{X}^m$
17: if $\|\vec{\boldsymbol{z}}\|{>}(\alpha{-}1){\vee}\|\vec{\boldsymbol{s}}\| > (\tau_2{-}\beta\tau{-}\beta^2\tau_1)$:
18:     go to Step 3 ▷ Rejection Sampling
19: $\vec{s}_1' = \sum_{i=0}^{L'-1} 2^i \vec{\boldsymbol{z}}_i \in \mathbb{X}$
20: $\vec{\boldsymbol{z}}' = [z_i \mathsf{rot}(1, i)]_{i=0}^{L'-1} \in \mathbb{X}^{L'}$
21: $\vec{s}_2'{=}\vec{x}_1 \sum_{i=0}^{L'-1} z_i'(z_i' - \vec{x}_2 \mathsf{rot}(1, i)){\in}\mathbb{X}$
22: if $\|\vec{s}_1', \vec{s}_2'\| > \gamma$ : go to Step 3
23: ▷ Hints for $\vec{\boldsymbol{t}}_2$
24: $\vec{\boldsymbol{t}}_2'{=}\texttt{highbits}_{\gamma'}((\vec{\boldsymbol{K}}\vec{\boldsymbol{s}}{+}\vec{\boldsymbol{V}}\vec{s}_1'{+}\vec{\boldsymbol{R}}\vec{s}_2'$
25:         $-\vec{x}_2(\vec{\boldsymbol{c}}{+}\vec{x}_1\mathtt{up}_{p_1}(\vec{\boldsymbol{t}}_1))){\in}\mathbb{X}_q^n$
26: $\vec{\boldsymbol{h}} = \texttt{hints}(\vec{\boldsymbol{t}}_2', \vec{\boldsymbol{t}}_2)$
27: if $\vec{\boldsymbol{h}} = \perp$: go to go to Step 3
28: return $\pi = (\vec{\boldsymbol{z}}, \vec{\boldsymbol{s}}, \vec{\boldsymbol{t}}_1, \vec{x}_2, \vec{\boldsymbol{h}})$

▷ Range Proof Verification
$\underline{\texttt{CIP}_{\texttt{SIS}}.\texttt{ver}(\texttt{range}_{L'}; \vec{\boldsymbol{c}}, \pi) :}$
$(\vec{\boldsymbol{z}}, \vec{\boldsymbol{s}}, \vec{\boldsymbol{t}}_1, \vec{x}_2, \vec{\boldsymbol{h}}) := \pi$
if $\|\vec{\boldsymbol{z}}\| > \alpha$ : return 0
$\vec{x}_1 = \mathsf{hash}(\vec{\boldsymbol{c}}, \vec{\boldsymbol{t}}_1) \in \mathcal{C}_\beta^N$
$\vec{s}_1 = \sum_{i=0}^{L'-1} 2^i \vec{\boldsymbol{z}}_i \in \mathbb{X}$
$\vec{\boldsymbol{z}}' = [z_i \mathsf{rot}(1, i)]_{i=0}^{L'-1}] \in \mathbb{X}^{L'}$
$\vec{s}_2{=}\vec{x}_1 \sum_{i=0}^{L'-1} z_i'(z_i' - \vec{x}_2 \mathsf{rot}(1, i)){\in}\mathbb{X}$
▷ Recreate $\vec{\boldsymbol{t}}_2$
$\vec{\boldsymbol{t}}_2'{=}\texttt{highbits}_{\gamma'}(\vec{\boldsymbol{K}}\vec{\boldsymbol{s}}{+}\vec{\boldsymbol{V}}\vec{s}_1{+}\vec{\boldsymbol{R}}\vec{s}_2$
    $-\vec{x}_2(\vec{\boldsymbol{c}}{+}\vec{x}_1\mathtt{up}_{p_1}(\vec{\boldsymbol{t}}_1))){\in}\mathbb{X}_q^n$
$\vec{\boldsymbol{t}}_2 = \texttt{use\_hints}(\vec{\boldsymbol{t}}_2', \vec{\boldsymbol{h}})$
return $\vec{\boldsymbol{t}}_2 \neq \perp \wedge \vec{x}_2 \overset{?}{=} \mathsf{hash}(\vec{x}_1, \vec{\boldsymbol{t}}_2)$
    $\bigwedge \|\vec{\boldsymbol{s}}\| \leq \tau_2 \wedge \|\vec{s}_1, \vec{s}_2\| \leq \gamma$
```

The most primitive operations are additions and subtractions. Additions and subtractions are inherently **intra-private** computations, and input confidential integers are sufficient to prove the relation. Hence, these proofs only check the malleability to prevent the other party from changing the inputs. Note that addition and subtraction do not guarantee the range of the inputs or the outputs.

```
$\triangleright$ Addition and Subtraction
$\underline{\mathtt{CIP}_{\mathtt{DLP}}.\mathtt{prove}(\mathtt{add/sub}; \pi_0 = \{c_0, c_1, c_2\})}$
if $c_0 \neq c_1 \times c_2^{\pm 1} \in \mathbb{G}$:
   return $\perp$
return $\pi = \mathtt{hash}(c_0, c_1, c_2) \in \mathbb{Z}_q^+$
$\triangleright$ Argument Verification
$\underline{\mathtt{CIP}_{\mathtt{DLP}}.\mathtt{ver}(\mathtt{add/sub}; \pi_0 = \{c_0, c_1, c_2\}, \pi)}$
return $c_0{=}c_1 \times c_2^{\pm 1} \wedge \pi \overset{?}{=} \mathtt{hash}(c_0, c_1, c_2)$
```

```
$\triangleright$ Addition and Subtraction
$\underline{\mathtt{CIP}_{\mathtt{SIS}}.\mathtt{prove}(\mathtt{add/sub}; \pi_0 = \{\vec{\boldsymbol{c}}_0, \vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2\})}$
if $\vec{\boldsymbol{c}}_0 \neq \mathtt{up}_{\gamma'}(\mathtt{highbits}_{\gamma'}(\vec{\boldsymbol{c}}_1 \pm \vec{\boldsymbol{c}}_2))$:
   return $\perp$
return $\pi = \mathtt{hash}(\vec{\boldsymbol{c}}_0, \vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2) \in \mathcal{C}_\beta^N$
$\triangleright$ Argument Verification
$\underline{\mathtt{CIP}_{\mathtt{SIS}}.\mathtt{ver}(\mathtt{add/sub}; \pi_0 = \{\vec{\boldsymbol{c}}_0, \vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2\}, \pi)}$
return $\vec{\boldsymbol{c}}_0 = \mathtt{up}_{\gamma'}(\mathtt{highbits}_{\gamma'}(\vec{\boldsymbol{c}}_1 \pm \vec{\boldsymbol{c}}_2))$
   $\wedge \pi \overset{?}{=} \mathtt{hash}(\vec{\boldsymbol{c}}_0, \vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2)$
```

Our multiplication proofs check $v_0 = v_1 v_2 \pmod{q(> 2^{2+1})}$ in CIP-DLP and $v_0 = v_1 v_2$ in CIP-SIS for the given v_0, v_1, v_2.

```
$\triangleright$ Proving Confidential Multiplication
$\underline{\mathtt{CIP}_{\mathtt{DLP}}.\mathtt{prove}(\mathtt{mul}; \pi_0 = \{c_0, c_1, c_2\}}$
      $\underline{\boldsymbol{k} = \{v_0, k_0, v_1, k_1, v_2, k_2\}):}$
if $v_0{\neq}v_1 v_2 \pmod q \vee$
      $v_0, v_1, v_2 {\notin} (-2^L, 2^L)$ : return $\perp$
$k_1' \xleftarrow{\$} \mathbb{Z}_q^+$; $k_2' \xleftarrow{\$} \mathbb{Z}_q^+$
$a_1 \xleftarrow{\$} \mathbb{Z}_q^+$; $a_2 \xleftarrow{\$} \mathbb{Z}_q^+$
$t_1 = g^{k_1'} h^{v_1 a_1 + v_2 a_2} \in \mathbb{G}$
$\triangleright$ Challenge 0
$x_0 = \mathsf{hash}(c_0, c_1, c_2, t_1) \in \mathbb{Z}_q^+$
$\triangleright$ Challenge 1
$x_1 = \mathsf{hash}(x_0) \in \mathbb{Z}_q^+$ $\triangleright$ Challenge 2
$t_2 = g^{k_2'} h^{a_1 a_2 + x_0 a_1 + x_1 a_2} \in \mathbb{G}$
$x_2 = \mathsf{hash}(x_1, t_2) \in \mathbb{Z}_q^+$
$z_1 = a_1 + x_2 v_1$; $z_2 = a_2 + x_2 v_2 \in \mathbb{Z}_q^+$
$s{=}x_2^2 k_0 + x_2(k_1'{+}x_0 k_1{+}x_1 k_2) + k_2' \in \mathbb{Z}_q^+$
return $\pi = (z_1, z_2, s, t_1, t_2)$
```

```
$\triangleright$ Verifying Confidential Multiplication
$\underline{\mathtt{CIP}_{\mathtt{DLP}}.\mathtt{ver}(\mathtt{mul}; \pi_0 = \{c_0, c_1, c_2\}, \pi)}$
$(z_1, z_2, s, t_1, t_2) := \pi$
$x_0 = \mathsf{hash}(c_0, c_1, c_2, t_1) \in \mathbb{Z}_q^+$
$x_1 = \mathsf{hash}(x_0) \in \mathbb{Z}_q^+$
$x_2 = \mathsf{hash}(x_1, t_2) \in \mathbb{Z}_q^+$
return $g^s h^{z_1 z_2 + x_0 z_1 + x_1 z_2} \overset{?}{=} c_0^{(x_2)^2}$
            $\times (t_1 \times c_1^{x_0} \times c_2^{x_1})^{x_2} \times t_2 \in \mathbb{G}$
```

```
    $\triangleright$ Proving Confidential Multiplication
 1: $\underline{\mathtt{CIP}_{\mathtt{SIS}}.\mathtt{prove}(\mathtt{mul}; \pi_0 = \{\vec{\boldsymbol{c}}_0, \vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2\}}$
 2:       $\underline{\boldsymbol{k} = \{v_0, \vec{\boldsymbol{k}}_0, v_1, \vec{\boldsymbol{k}}_1, v_2, \vec{\boldsymbol{k}}_2\}):}$
 3: if $v_0{\neq}v_1 v_2 \vee v_0, v_1, v_2 {\notin} (-2^L, 2^L)$:
 4:       return $\perp$
 5: $\vec{\boldsymbol{k}}_1' \xleftarrow{\$} [-\tau_1, \tau_1]^{m \times N}$
 6: $\vec{\boldsymbol{k}}_2' \xleftarrow{\$} [-\tau_2, \tau_2]^{m \times N}$
 7: $\triangleright$ Lazy Sampling $w = 2^L{-}1$
 8: $\vec{a}_1 \xleftarrow{\$} [-(\alpha_1{-}(w{-}v_1)), (\alpha_1{-}(w{-}v_1))]^N {\in} \mathbb{X}$
 9: $\vec{a}_2 \xleftarrow{\$} [-(\alpha_1{-}(w{-}v_2)), (\alpha_1{-}(w{-}v_2))]^N {\in} \mathbb{X}$
10: $\vec{\boldsymbol{t}}_1 = \mathtt{highbits}_{p_1}(\vec{\boldsymbol{V}}(v_1 \vec{a}_2 + v_2 \vec{a}_1)$
11:       $+ \vec{\boldsymbol{K}} \vec{\boldsymbol{k}}_1') \in \mathbb{X}_q^n$
12: $\triangleright$ Challenge 0
13: $\vec{x}_0 = \mathsf{hash}(\vec{\boldsymbol{c}}_0, \vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2, \vec{\boldsymbol{t}}_1) \in \mathcal{C}_\beta^N$
14: $\triangleright$ Challenge 1
15: $\vec{x}_1 = \mathsf{hash}(\vec{x}_0) \in \mathcal{C}_\beta^N$
16: $\vec{a}' = \vec{a}_1 \vec{a}_2 + \vec{x}_0 \vec{a}_1 + \vec{x}_1 \vec{a}_2 \in \mathbb{X}$
17: $\vec{\boldsymbol{t}}_2 = \mathtt{highbits}_{\gamma'}(\vec{\boldsymbol{V}} \vec{a}' + \vec{\boldsymbol{K}} \vec{\boldsymbol{k}}_2') \in \mathbb{X}_q^n$
18: $\triangleright$ Challenge 2
19: $\vec{x}_2 = \mathsf{hash}(\vec{x}_1, \vec{\boldsymbol{t}}_2) \in \mathcal{C}_\beta^N$
20: $\vec{z}_1 = v_1 \vec{x}_2 + \vec{a}_1$, $\vec{z}_2 = v_2 \vec{x}_2 + \vec{a}_2 \in \mathbb{X}$
21: $\vec{\boldsymbol{s}}{=}\vec{x}_2^2 \vec{\boldsymbol{k}}_0 {+} \vec{x}_2(\vec{\boldsymbol{k}}_1' {+} \vec{x}_0 \vec{\boldsymbol{k}}_1 {+} \vec{x}_1 \vec{\boldsymbol{k}}_2) {+} \vec{\boldsymbol{k}}_2' \in \mathbb{X}^m$
22: $\vec{s}_1 = (\vec{z}_1 \vec{z}_2 + \vec{x}_0 \vec{z}_1 + \vec{x}_1 \vec{z}_2) \in \mathbb{X}$
23: if $\|\vec{s}_1\| > \gamma \vee \|\vec{z}_1\| > (\alpha - (2^L - 1))$
24:    $\vee \|\vec{z}_2\| > (\alpha - (2^L - 1)) \vee$
25:    $\|\vec{\boldsymbol{s}}\| {>} (\tau_2 {-} 3\beta^2 \tau {-} \beta \tau_1)$ : go to Step 5
    $\triangleright$ Hints for $\vec{\boldsymbol{t}}_2$
26: $\vec{\boldsymbol{t}}_2' = \mathtt{highbits}_{\gamma'}(\vec{\boldsymbol{K}} \vec{\boldsymbol{s}} + \vec{\boldsymbol{V}} \vec{s}_1 - \vec{x}_2^2 \vec{\boldsymbol{c}}_0$
27:    $-\vec{x}_2(\mathtt{up}_{p_1}(\vec{\boldsymbol{t}}_1) + \vec{x}_0 \vec{\boldsymbol{c}}_1 + \vec{x}_1 \vec{\boldsymbol{c}}_2)) \in \mathbb{X}_q^n$
28: $\vec{\boldsymbol{h}} = \mathtt{hints}(\vec{\boldsymbol{t}}_2', \vec{\boldsymbol{t}}_2)$
29: if $\vec{\boldsymbol{h}} = \perp$: go to go to Step 5
30: return $\pi = (\vec{z}_1, \vec{z}_2, \vec{\boldsymbol{s}}, \vec{\boldsymbol{t}}_1, \vec{x}_2, \vec{\boldsymbol{h}})$
```

```
$\triangleright$ Verifying Confidential Multiplication
$\underline{\mathtt{CIP}_{\mathtt{SIS}}.\mathtt{ver}(\mathtt{mul}; \pi_0 = \{\vec{\boldsymbol{c}}_0, \vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2\}, \pi)}$
$(\vec{z}_1, \vec{z}_2, \vec{\boldsymbol{s}}, \vec{\boldsymbol{t}}_1, \vec{x}_2, \vec{\boldsymbol{h}}) := \pi$
$\vec{x}_0 = \mathsf{hash}(\vec{\boldsymbol{c}}_0, \vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2, \vec{\boldsymbol{t}}_1) \in \mathcal{C}_\beta^N$
$\vec{x}_1 = \mathsf{hash}(\vec{x}_0) \in \mathcal{C}_\beta^N$
$\vec{s}_1 = (\vec{z}_1 \vec{z}_2 + \vec{x}_0 \vec{z}_1 + \vec{x}_1 \vec{z}_2)) \in \mathbb{X}$
if $\|[\vec{z}_1, \vec{z}_2]\| > \alpha \vee \|\vec{\boldsymbol{s}}\| > \tau_2 \vee \|\vec{s}_1\| > \gamma$:
   return 0
$\vec{\boldsymbol{t}}_2 = \mathtt{highbits}_{\gamma'}(\vec{\boldsymbol{K}} \vec{\boldsymbol{s}} + \vec{\boldsymbol{V}} \vec{s}_1 - \vec{x}_2^2 \vec{\boldsymbol{c}}_0$
   $-\vec{x}_2(\mathtt{up}_{p_1}(\vec{\boldsymbol{t}}_1) + \vec{x}_0 \vec{\boldsymbol{c}}_1 + \vec{x}_1 \vec{\boldsymbol{c}}_2)) \in \mathbb{X}_q^n$
$\vec{\boldsymbol{t}}_2 = \mathtt{use\_hints}(\vec{\boldsymbol{t}}_2', \vec{\boldsymbol{h}})$ $\triangleright$ Recreate $\vec{\boldsymbol{t}}_2$
return $\vec{\boldsymbol{t}}_2 \neq \perp \wedge \vec{x}_2 \overset{?}{=} \mathsf{hash}(\vec{x}_1, \vec{\boldsymbol{t}}_2)$
```

Before the division; we will explain conditionals. In "equal" statements, the prover shows that the subtraction of the two commitments produces a commitment of a zero. However, checking the "not equal" requires some work. Assume we want to check that $x \overset{?}{\neq} 3$. When $x \neq 3$, $-(x-(3+1))((3-1)-x)$ will be always positive. We can check it by asking to compute a range proof for $-(x-(3+1))((3-1)-x)$. Our "smaller or larger than or equal" statements subtract confidential integers and asks to compute range proofs accordingly. For example, if $\vec{\boldsymbol{c}}_1$'s committed integers are larger than $\vec{\boldsymbol{c}}_2$'s committed integers, then the prover can create a non-negative range proof for $\vec{\boldsymbol{c}}_1 - \vec{\boldsymbol{c}}_2$. We state our confidential conditionals below.

```
$\triangleright$ Confidential Equal Argument
$\underline{\texttt{CIP}_{\texttt{DLP}}.\texttt{prove}(\texttt{eq}; \pi_0 = \{c_1, c_2\}}$
        $\underline{\boldsymbol{k} = \{v_1, k_1, v_1, k_2\}):}$
if $v_0 \neq v_1 \pmod{q}$ : return $\perp$
$k' \xleftarrow{\$} \mathbb{Z}_q^+;\ t = g^{k'} \in \mathbb{G}$
$x$=$\texttt{hash}(c_1, c_2, t)$
return $\pi = (s = (k' + x(k_1 - k_2)) \in \mathbb{Z}_q^+, t)$

$\triangleright$ Confidential Equal Verification
$\underline{\texttt{CIP}_{\texttt{DLP}}.\texttt{ver}(\texttt{eq}; \pi_0 = \{c_1, c_2\}, \pi = (s, t))}$
return $g^s \overset{?}{=} t \times (c_1 c_2^{-1})^{\texttt{hash}(c_1, c_2, t)} \in \mathbb{G}$

$\triangleright$ Confidential Not Equal Proving
$\underline{\texttt{CIP}_{\texttt{DLP}}.\texttt{prove}(\texttt{neq}; \pi_0 = \{c_1, c_2\}}$
        $\underline{\boldsymbol{k} = \{v_1, k_1, v_1, k_2\}):}$
if $v_1 = v_2 \pmod{q}$ : return $\perp$
$v_0 = -(v_1 - (v_2 + 1))((v_2 - 1) - v_1)$
$(k_0, c_0) := \texttt{CIP}_{\texttt{DLP}}.\texttt{cint}(v_0)$
$\pi' = \texttt{CIP}_{\texttt{DLP}}.\texttt{prove}\big(\texttt{mul};$
    $\pi_0 = \{c_0, (c_1)^{-1} c_2 e, (c_2 (h c_1)^{-1}\}$
    $\boldsymbol{k} = \{v_0, k_0, -v_1 + v_2 + 1, -k_1 + k_2,$
        $((v_2 - 1) - v_1), (k_2 - k_1)\}\big)$
$\pi'' = \texttt{CIP}_{\texttt{DLP}}.\texttt{prove}\big(\texttt{range}_{2L+5};$
        $c_0, \{v_0, k_0\}\big)$ $\triangleright$ $2L + 1$ not $L$
return $\pi = (c_0, \pi', \pi'')$

$\triangleright$ Confidential Not Equal Verification
$\underline{\texttt{CIP}_{\texttt{DLP}}.\texttt{ver}(\texttt{neq}; \pi_0 = \{c_1, c_2\}, \pi)}$
$(c_0, \pi', \pi'') := \pi$
$\pi'_0 = \{c_0, (c_1)^{-1} c_2 e, (c_2 (h c_1)^{-1}\}$
return $\texttt{CIP}_{\texttt{DLP}}.\texttt{ver}\big(\texttt{mul}; \pi'_0, \pi'\big)$
    $\wedge \texttt{CIP}_{\texttt{DLP}}.\texttt{ver}\big(\texttt{range}_{2L+5}; \{c_0\}, \pi''\big)$

$\triangleright$ Confidential Less Than and Equal
$\underline{\texttt{CIP}_{\texttt{DLP}}.\texttt{prove}(\texttt{leq}; \pi_0 = \{c_1, c_2\}}$
        $\underline{\boldsymbol{k} = \{v_1, k_1, v_1, k_2\}):}$
if $v_1 > v_2$ : return $\perp$
$\pi = \big(\texttt{hash}(c_1, c_2), \texttt{CIP}_{\texttt{DLP}}.\texttt{prove}\big(\texttt{range}_L;$
    $\pi_0 = \{c_2 (c_1)^{-1}\}, \boldsymbol{k} = \{v_2 - v_1, k_2 - k_1\}\big)\big)$
return $\pi$

$\triangleright$ Confidential Less Than and Equal
$\underline{\texttt{CIP}_{\texttt{DLP}}.\texttt{ver}(\texttt{leq}; \pi_0 = \{c_1, c_2\}, \pi = (x, \pi'))}$
return $\texttt{CIP}_{\texttt{SIS}}.\texttt{ver}\big(\texttt{range}_L; c_2 (c_1)^{-1}, \pi'\big)$
    $\wedge x = \texttt{hash}(c_1, c_2)$

$\triangleright$ Confidential Greater Than and Equal
$\underline{\texttt{CIP}_{\texttt{DLP}}.\texttt{prove}(\texttt{geq}; \pi_0 = \{c_1, c_2\}}$
        $\underline{\boldsymbol{k} = \{v_1, k_1, v_1, k_2\}):}$
if $v_1 < v_2$ : return $\perp$
$\pi = \big(\texttt{hash}(c_1, c_2), \texttt{CIP}_{\texttt{DLP}}.\texttt{prove}\big(\texttt{range}_L;$
    $\pi_0 = \{c_1 (c_2)^{-1}\}, \boldsymbol{k} = \{v_1 - v_2, k_1 - k_2\}\big)\big)$
return $\pi$

$\triangleright$ Confidential Greater Than and Equal
$\underline{\texttt{CIP}_{\texttt{DLP}}.\texttt{ver}(\texttt{geq}; \pi_0 = \{c_1, c_2\}, \pi = (x, \pi'))}$
return $\texttt{CIP}_{\texttt{SIS}}.\texttt{ver}\big(\texttt{range}_L; c_1 (c_2)^{-1}, \pi'\big)$
    $\wedge x = \texttt{hash}(c_1, c_2)$
```

```
 1: $\triangleright$ Confidential Equal Argument
 2: $\underline{\texttt{CIP}_{\texttt{SIS}}.\texttt{prove}(\texttt{eq}; \pi_0 = \{\vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2\}}$
 3:         $\underline{\boldsymbol{k} = \{v_1, \vec{\boldsymbol{k}}_1, v_2, \vec{\boldsymbol{k}}_2\}):}$
 4: if $v_1 \neq v_2$ : return $\perp$
 5: $\vec{\boldsymbol{k}}' \leftarrow [-\tau_3, \tau_3]^{m \times N}$ ;
 6: $\vec{\boldsymbol{t}} = \mathsf{highbits}_{\gamma'}(\vec{\boldsymbol{K}} \vec{\boldsymbol{k}}') \in \mathbb{X}_q^n$
 7: $\vec{x} = \mathsf{hash}(\vec{\boldsymbol{c}}_0, \vec{\boldsymbol{c}}_1, \vec{\boldsymbol{t}}) \in \mathcal{C}_\beta^N$;
 8: $\vec{\boldsymbol{s}} = \vec{x}(\vec{\boldsymbol{k}}_1 - \vec{\boldsymbol{k}}_2) + \vec{\boldsymbol{k}}' \in \mathbb{X}^m$
 9: if $\|\vec{\boldsymbol{s}}\| > (\tau_3 - 2\beta\tau)$ : go to Step 5
10: $\triangleright$ Hints for $\vec{\boldsymbol{t}}$
11: $\vec{\boldsymbol{t}}' = \mathsf{highbits}_{\gamma'}(\vec{\boldsymbol{K}} \vec{\boldsymbol{s}} - \vec{x}(\vec{\boldsymbol{c}}_1 - \vec{\boldsymbol{c}}_2)) \in \mathbb{X}_q^n$
12: $\vec{\boldsymbol{h}} = \mathsf{hints}(\vec{\boldsymbol{t}}', \vec{\boldsymbol{t}})$
13: if $\vec{\boldsymbol{h}} = \perp$: go to Step 5
14: return $\pi = (\vec{\boldsymbol{s}}, \vec{x}, \vec{\boldsymbol{h}})$
```

```
▷ Confidential Equal Verification
$\underline{\texttt{CIP}_{\texttt{SIS}}\texttt{.ver}(\texttt{eq}; \pi_0 = \{\vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2\}, \pi = (\vec{\boldsymbol{s}}, \vec{x}, \vec{\boldsymbol{h}}))}$
▷ Recreate $\vec{\boldsymbol{t}}$
$\vec{\boldsymbol{t}}' = \mathsf{highbits}_{\gamma'}(\vec{\boldsymbol{K}}\vec{\boldsymbol{s}} - \vec{x}(\vec{\boldsymbol{c}}_1 - \vec{\boldsymbol{c}}_2)) \in \mathbb{X}_q^n$
$\vec{\boldsymbol{t}} = \mathsf{use\_hints}(\vec{\boldsymbol{t}}', \vec{\boldsymbol{h}})$
return $\vec{\boldsymbol{t}} \neq \perp \wedge \vec{x} \overset{?}{=} \mathsf{hash}(\vec{\boldsymbol{c}}_0, \vec{\boldsymbol{c}}_1, \vec{\boldsymbol{t}}) \wedge \|\vec{\boldsymbol{s}}\| \leq \gamma$
```

```
▷ Confidential Not Equal Argument
$\underline{\texttt{CIP}_{\texttt{SIS}}\texttt{.prove}(\texttt{neq}; \pi_0 = \{\vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2\}}$
        $\underline{\boldsymbol{k} = \{v_1, \vec{\boldsymbol{k}}_1, v_2, \vec{\boldsymbol{k}}_2\}):}$
if $v_1 = v_2$ : return $\perp$
$\vec{\boldsymbol{e}} = \vec{\boldsymbol{V}}\mathsf{rot}(1, 0) \in \mathbb{X}_q^n$
$v_0 = -(v_1 - (v_2 + 1))((v_2 - 1) - v_1)$
$(\vec{\boldsymbol{k}}_0, \vec{\boldsymbol{c}}_0) := \texttt{CIP}_{\texttt{SIS}}\texttt{.cint}(v_0)$
$\pi' = \texttt{CIP}_{\texttt{SIS}}\texttt{.prove}\big(\texttt{mul};$
    $\pi_0 = \{\vec{\boldsymbol{c}}_0, -\vec{\boldsymbol{c}}_1 + \vec{\boldsymbol{c}}_2 + e, (\vec{\boldsymbol{c}}_2 - e) - \vec{\boldsymbol{c}}_1\}$
    $\boldsymbol{k} = \{v_0, \vec{\boldsymbol{k}}_0, -v_1 + v_2 + 1, -\vec{\boldsymbol{k}}_1 + \vec{\boldsymbol{k}}_2,$
            $((v_2 - 1) - v_1), (\vec{\boldsymbol{k}}_2 - \vec{\boldsymbol{k}}_1)\}\big)$
$\pi'' = \texttt{CIP}_{\texttt{SIS}}\texttt{.prove}\big(\texttt{range}_{2L+5};$
        $\vec{\boldsymbol{c}}_0, \{v_0, \vec{\boldsymbol{k}}_0\}\big)$ ▷ $2L + 1$ not $L$
return $\pi = (\vec{\boldsymbol{c}}_0, \pi', \pi'')$
```

```
▷ Confidential Not Equal Verification
$\underline{\texttt{CIP}_{\texttt{SIS}}\texttt{.ver}(\texttt{neq}; \pi_0 = \{\vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2\}, \pi)}$
$(\vec{\boldsymbol{c}}_0, \pi', \pi'') := \pi$
$\pi_0' = \{\vec{\boldsymbol{c}}_0, -\vec{\boldsymbol{c}}_1 + \vec{\boldsymbol{c}}_2 + e, (\vec{\boldsymbol{c}}_2 - e) - \vec{\boldsymbol{c}}_1\}$
return $\texttt{CIP}_{\texttt{SIS}}\texttt{.ver}\big(\texttt{mul}; \pi_0', \pi'\big)$
    $\wedge \texttt{CIP}_{\texttt{SIS}}\texttt{.ver}\big(\texttt{range}_{2L+5}; \{\vec{\boldsymbol{c}}_0\}, \pi''\big)$
```

```
▷ Confidential Less Than and Equal
$\underline{\texttt{CIP}_{\texttt{SIS}}\texttt{.prove}(\texttt{leq}; \pi_0 = \{\vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2\}}$
        $\underline{\boldsymbol{k} = \{v_1, \vec{\boldsymbol{k}}_1, v_2, \vec{\boldsymbol{k}}_2\}):}$
if $v_1 > v_2$ : return $\perp$
$\pi = \big(\texttt{hash}(\pi_0), \texttt{CIP}_{\texttt{SIS}}\texttt{.prove}\big(\texttt{range}_L;$
    $\pi_0 = \{\vec{\boldsymbol{c}}_2 - \vec{\boldsymbol{c}}_1\}, \boldsymbol{k} = \{v_2 - v_1, \vec{\boldsymbol{k}}_2 - \vec{\boldsymbol{k}}_1\}\big)\big)$
return $\pi$
```

```
▷ Confidential Less Than and Equal
$\underline{\texttt{CIP}_{\texttt{SIS}}\texttt{.ver}(\texttt{leq}; \pi_0 = \{\vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2\}, \pi = (\vec{x}, \pi'))}$
return $\texttt{CIP}_{\texttt{SIS}}\texttt{.ver}\big(\texttt{range}_L; \vec{\boldsymbol{c}}_2 - \vec{\boldsymbol{c}}_1, \pi'\big)$
    $\vec{x} = \texttt{hash}(\pi_0)$
```

```
▷ Confidential Greater Than and Equal
$\underline{\texttt{CIP}_{\texttt{SIS}}\texttt{.prove}(\texttt{geq}; \pi_0 = \{\vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2\}}$
        $\underline{\boldsymbol{k} = \{v_1, \vec{\boldsymbol{k}}_1, v_2, \vec{\boldsymbol{k}}_2\}):}$
if $v_1 < v_2$ : return $\perp$
$\pi = \big(\texttt{hash}(\pi_0), \texttt{CIP}_{\texttt{SIS}}\texttt{.prove}\big(\texttt{range}_L;$
    $\pi_0 = \{\vec{\boldsymbol{c}}_1 - \vec{\boldsymbol{c}}_2\}, \boldsymbol{k} = \{v_1 - v_2, \vec{\boldsymbol{k}}_1 - \vec{\boldsymbol{k}}_2\}\big)\big)$
return $\pi$
```

```
▷ Confidential Greater Than and Equal
$\underline{\texttt{CIP}_{\texttt{SIS}}\texttt{.ver}(\texttt{geq}; \pi_0 = \{\vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2\}, \pi = (\vec{x}, \pi'))}$
return $\texttt{CIP}_{\texttt{SIS}}\texttt{.ver}\big(\texttt{range}_L; \vec{\boldsymbol{c}}_1 - \vec{\boldsymbol{c}}_2, \pi'\big)$
    $\vec{x} = \texttt{hash}(\pi_0)$
```

We provide two divisions: unsigned integer division and signed integer division. Let v_0, v_1, v_2, and v_3 be such that $v_1 = v_0 v_2 + v_3$. If v's is unsigned (or non-negative) the remainder v_3 should be less than the divisor v_2. When the numbers are signed, we instead check $v_3^2 < v_2^2$ and $v_3 \geq 0$ (in CIP-DLP, v_3^2 and v_2^2 are in $[0, q]$ but they will not overflow since $q > 2^{2L+5}$). We use these facts to compute zero-knowledge proofs for signed and unsigned divisions. Our protocols are stated in Appendix A.

Note that CIP-DLP's "multi-party equal" proofs were explained in Figure 1. Finally, we explain "multi-party equal" for CIP-SIS below. Here, a participant $\mathcal{P}_1$ has a confidential integer $\vec{\boldsymbol{c}}_1$ of $(v, \vec{\boldsymbol{k}}_1)$, and another participant $\mathcal{P}_2$ has $\vec{\boldsymbol{c}}_2$ of $(v, \vec{\boldsymbol{k}}_2)$. None of them wants to share their secret keys but they want show that the integers are the same. They compute a "multi-party equal" proof as follows.

- Each participant i picks a secret $\vec{\boldsymbol{k}}_i' \in [-\tau_3/2\beta, \tau_3/2\beta]^{m \times N}$ and computes a confidential integer of zero, $\vec{\boldsymbol{t}}_i = \texttt{highbits}_{\gamma'-2\beta}(\vec{\boldsymbol{K}}\vec{\boldsymbol{k}}_i') \in \mathbb{X}_q^n$. They share each $\vec{\boldsymbol{t}}_i$. Let $\boldsymbol{t} = (\vec{\boldsymbol{t}}_1, \vec{\boldsymbol{t}}_2)$ be the shared values.
- They both compute; $\vec{\boldsymbol{t}} = \texttt{highbits}_{\gamma'}(\texttt{hash}(\boldsymbol{t}, \vec{\boldsymbol{t}}_1)\vec{\boldsymbol{t}}_1 - \texttt{hash}(\boldsymbol{t}, \vec{\boldsymbol{t}}_2)\vec{\boldsymbol{t}}_2) \in \mathbb{X}_q^n$.
- Then each participant i computes a challenge $\vec{x} = \texttt{hash}(\vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2, \vec{\boldsymbol{t}}) \in \mathcal{C}_\beta^N$ and $\vec{\boldsymbol{s}}_i = \texttt{hash}(\boldsymbol{t}, \vec{\boldsymbol{t}}_i)\vec{\boldsymbol{k}}_i' + \vec{x}\vec{\boldsymbol{k}}_i \in \mathbb{X}^m$.

- However, before sharing them, they perform rejection sampling. If $\|\vec{\boldsymbol{s}}_i\| > [\tau_3/2 - \tau\beta]$, both restart the protocol. Otherwise, they share $\vec{\boldsymbol{s}}_i$ and compute $\vec{\boldsymbol{s}} = \vec{\boldsymbol{s}}_1 - \vec{\boldsymbol{s}}_2 \in \mathbb{X}^m$.
- Then compute hints for $\vec{\boldsymbol{t}}$ such that $\vec{\boldsymbol{h}} = \texttt{hints}(\texttt{hightbits}_{\gamma'}(\vec{\boldsymbol{K}}\vec{\boldsymbol{s}} - \vec{\boldsymbol{c}}_1 + \vec{\boldsymbol{c}}_2), \vec{\boldsymbol{t}})$.
- They compute the proof $\pi = (\vec{\boldsymbol{s}}, \vec{x}, \vec{\boldsymbol{h}})$ and send π to the verifier.

The verification works as follows. The verifier recomputes $\vec{\boldsymbol{t}}$ from hints such that $\vec{\boldsymbol{t}} = \texttt{use\_hint}(\texttt{hightbits}_{\gamma'}(\vec{\boldsymbol{K}}\vec{\boldsymbol{s}} - \vec{\boldsymbol{c}}_1 + \vec{\boldsymbol{c}}_2), \vec{\boldsymbol{h}}) \in \mathbb{X}_q^n$. If $\vec{x} \overset{?}{=} \texttt{hash}(\vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2, \vec{\boldsymbol{t}})$ and $\vec{\boldsymbol{t}} \neq \perp$, the verifier accepts the proof.

5 Security Realization

With these concrete protocols, we can state the following security theorem.

Theorem 3. *$\mathcal{R}$:$(\texttt{CIP}_{\texttt{DLP}}, \mathcal{A})$ and $\mathcal{R}$:$(\texttt{CIP}_{\texttt{SIS}}, \mathcal{A})$ realize $\mathcal{I}$:$(\{\mathcal{F}_{\texttt{CIP}}, \mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}\}, \mathcal{S}_{\mathcal{A}})$* if and only if *$\texttt{CIP}_{\texttt{DLP}}$ and $\texttt{CIP}_{\texttt{SIS}}$ are complete, DLP is hard for $(\mathbb{G}, q)$, and Approx-SIS is hard for $(n, m+1, q, \gamma, \gamma', N)$, respectively.*

Correctness of the Ideal System. Proving security in the UC framework has two major steps; identifying the proper ideal system and finding a real protocol that realizes the ideal function(s). UC realization stated in Theorem 4 shows whether we have come up with a proper ideal system or not before going to the second step. First, we define the trapdoors of simulated adversaries.

- In $\texttt{CIP}_{\texttt{DLP}}$, $\mathcal{S}_{\mathcal{A}}$ has a trapdoor $\mathcal{T}_{\texttt{DLP}}$ which solves the discrete logarithms of g, e.g., given $y \xleftarrow{\$} \mathbb{G}$, $\mathcal{S}(pp_{\lambda,L}, \mathcal{T}_{\texttt{DLP}}, \mathcal{L}, \pi)$ can get $k \in \mathbb{Z}_q^+$ such that $y = g^k \in \mathbb{G}$.
- $\texttt{CIP}_{\texttt{SIS}}$'s $\mathcal{S}_{\mathcal{A}}$ is given $\mathcal{T}_{\texttt{SIS}}$ to solve iMSIS of $\vec{\boldsymbol{K}}$ for τ, e.g., given $\vec{\boldsymbol{y}} \xleftarrow{\$} \mathbb{X}_q^n$, $\mathcal{S}(pp_{\lambda,L}, \mathcal{T}_{\texttt{SIS}}, \mathcal{L}, \pi)$ can sample $\vec{\boldsymbol{k}} \in \mathbb{X}^m$ such that $\vec{\boldsymbol{y}} = \vec{\boldsymbol{K}}\vec{\boldsymbol{k}} \in \mathbb{X}_q^n$ and $\|\vec{\boldsymbol{k}}\| \leq \tau$.

Note that, in real implementations, public parameters are securely generated and will not have these trapdoors, e.g., g and $\vec{\boldsymbol{K}}\|\vec{\boldsymbol{V}}$ will be hash extensions of some public seed strings to prevent intentional trapdoors.

Theorem 4 ("Only If" Directions). *$\mathcal{R}$: $(\texttt{CIP}, \mathcal{A})$ realizes an ideal system $\mathcal{I}$: $(\{\mathcal{F}_{\texttt{CIP}}, \mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}\}, \mathcal{S}_{\mathcal{A}})$* ***only if*** *$\texttt{CIP}$ is complete, binding, zero-knowledge, intra-private, and witness-extractable.*

Proof. The environment $\mathcal{E}$ can identify the real system over the ideal system if $\texttt{CIP}$ is not *complete* or honestly generated confidential integer/arguments are not always valid since $\mathcal{F}_{\texttt{CIP}}$ always considers previously generated confidential integer or arguments as valid. Thus, $\texttt{CIP}$ must be complete if $\mathcal{R}$ realizes $\mathcal{I}$.

Assume, $\texttt{CIP}$ is not binding or a p.p.t. adversary can find two different openings for the same confidential integer with more than non-negligible probability. The real system $\texttt{CIP}$ considers these non-binding confidential integers as valid while $\mathcal{F}_{\texttt{CIP}}$ considers them invalid. Therefore, $\texttt{CIP}$ must be at least computationally binding; or else, $\mathcal{E}$ correctly identifies the system it is interacting with.

$\mathcal{S}_{\mathcal{A}}$ sends simulated or properly generated proofs when the prover is corrupted. $\mathcal{S}_{\mathcal{A}}$ generates these simulated proofs using the trapdoor[3].

Therefore, $\mathcal{E}$ can identify the ideal system over the real system if `CIP` is not zero-knowledge or intra-private since only the ideal system replies with simulated proofs or transcripts to its cheating provers (Definition 8). Hence, `CIP` must be zero-knowledge and intra-private if it realizes $\{\mathcal{F}_{\texttt{CIP}}, \mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}\}$.

The ideal system and real system work differently for "cheating yet valid" proofs ("valid" according to the verification function) because the ideal system uses the extractor $\mathcal{K}$ to extract the hidden integers of π_0. $\mathcal{K}(pp_{\lambda,L}, \mathcal{T}, r; \pi_0, \pi)$ is also given a trapdoor, (i.e., $\mathcal{T} = \mathcal{T}_{\texttt{DLP}}$ or $\mathcal{T} = \mathcal{T}_{\texttt{SIS}}$) which does not exist or is unknown the real protocol [4]. Therefore, "cheating yet valid" proofs are considered invalid in the ideal system. If $\mathcal{E}$'s $\mathcal{A}$ adversary can create "cheating yet valid" proofs with some probability, $\mathcal{E}$ can identify the real system over the ideal system with the same probability due to $\mathcal{K}$'s responses. Hence, `CIP` must be simulation extractable; otherwise, $\mathcal{E}$ wins the game, and `CIP` does not realize $\{\mathcal{F}_{\texttt{CIP}}, \mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}\}$.

Therefore, `CIP` realizing $\{\mathcal{F}_{\texttt{CIP}}, \mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}\}$ means that `CIP` must be complete, binding, zero-knowledge, intra-private, and knowledge sound. Finally, we can conclude that Theorem 4 is valid. □

[3] For example, $\mathcal{S}(pp_{\lambda,L}, \mathcal{T}_{\texttt{SIS}}, \mathcal{L}, v)$ simulates DLP-Confidential integers as follows; $c \xleftarrow{\$} \mathbb{G}$ and returns (k, c) when $g^k = c(h^v)^{-1}$. Also, $\mathcal{S}(pp_{\lambda,L}, \mathcal{T}_{\texttt{SIS}}, \mathcal{L}, \pi)$ generates "equal" proofs as follows,

- picks random $c_1, c_2, t \xleftarrow{\$} \mathbb{G}$, computes $x := \texttt{hash}(c_1, c_2, t)$, and
- finds s such that $t \times (c_1 c_2)^x = g^s \in \mathbb{G}$.
- Then the simulated proof is (s, t) for simulated $\pi_0 = [c_1, c_2]$.

Similarly, in CIP-SIS, $\mathcal{S}(pp_{\lambda,L}, \mathcal{T}_{\texttt{SIS}}, \mathcal{L}, \pi)$ simulates "equal" proofs as follows,

- picks random $\vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2, \vec{\boldsymbol{t}}', \vec{\boldsymbol{h}} \xleftarrow{\$} \mathbb{X}_q^n$ such that $\|\vec{\boldsymbol{h}}\|_1 \leq \chi$ and $\|\vec{\boldsymbol{h}}\| = 1$,
- gets random $\vec{\boldsymbol{s}}' \in [-\tau_3 + \tau + 2\beta, \tau_3 - \tau - 2\beta]^{m \times N}$
- computes $[\vec{\boldsymbol{c}}_i = \texttt{up}_{p_1}(\texttt{highbits}_{p_1}(\vec{\boldsymbol{c}}_i))]_{i=1}^2$ and $\vec{\boldsymbol{t}} = \texttt{highbits}_{\gamma'}(\vec{\boldsymbol{t}}' + \vec{\boldsymbol{K}}\vec{\boldsymbol{s}}') + \vec{\boldsymbol{h}}$,
- computes $\vec{x} := \texttt{hash}(\vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2, \vec{\boldsymbol{t}})$, and
- finds $\vec{\boldsymbol{k}} \in \mathbb{X}^m$ such that $\vec{\boldsymbol{K}}\vec{\boldsymbol{k}} = \vec{\boldsymbol{t}}' - \vec{x}(\vec{\boldsymbol{c}}_1 - \vec{\boldsymbol{c}}_2)) \in \mathbb{X}_q^n$ and $\|\vec{\boldsymbol{k}}\| \leq \tau$.
- The simulated proof is $\pi = (\vec{x}, \vec{\boldsymbol{s}} = (\vec{\boldsymbol{s}}' + \vec{\boldsymbol{k}}), \vec{\boldsymbol{h}})$ for simulated $\pi_0 = [\vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2]$.

Likewise, $\mathcal{S}(pp_{\lambda,L}, \mathcal{T}_{\texttt{SIS}}, \mathcal{L}, \pi)$ can simulate any primitive relation. Hence the language only has primitive relations and their composed relations, it can simulate the entire language $\mathcal{L}$ at random. Not only the proofs, $\mathcal{S}(pp_{\lambda,L}, \mathcal{T}_{\texttt{SIS}}, \mathcal{L}, \pi)$ can also simulate transcripts, e.g., the simulated transcripts (t_1, t_2) will be randomly picked in "multiparty equal" proofs.

[4] Let us show an example of extracting values for "equal" proofs in CIP-DLP.

- Let $\pi_0 := [c_1, c_2]$ and the proof $\pi = (s, t)$.
- $\mathcal{K}$ gets k' of t when $t = g^{k'} \in \mathbb{G}$ and computes $k := (s - k')x^{-1} \in \mathbb{Z}_q^+$.
- Then $\mathcal{K}$ tries to find k_1 and k_2 such that $k = (k_1 - k_2)$ and $c_1 = g^{k_1} h^v$ and $c_2 = g^{k_2} h^v$ for some $v \in (0, q)$.
- If $\mathcal{K}$ returns (v, v) if it find the values; otherwise, sends $\perp$.

Informally, Theorem 4 proves that we have defined the ideal system: $\mathcal{F}_{\texttt{CIP}}$, $\mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}$, and $\mathcal{S}_{\mathcal{A}}$, *properly* to capture the expected security properties of `CIP`.

Theorem 5. *Confidential integers of* $\texttt{CIP}_{\texttt{DLP}}$ *and* $\texttt{CIP}_{\texttt{SIS}}$ *are hiding and binding if DLP and Approx-SIS is hard, respectively, for the same public parameters. (see [46] and [18]).*

Theorem 6. *Bullet-proof range proofs,* `BRP`*, is hiding, zero-knowledge, and simulation extractable when DLP is hard (see [14]).*

Theorem 7 ("If" Direction). $\mathcal{R} : (\texttt{CIP}_{\texttt{DLP}}, \mathcal{A})$ *and* $\mathcal{R} : (\texttt{CIP}_{\texttt{SIS}}, \mathcal{A})$ *realize* $\mathcal{I} : (\{\mathcal{F}_{\texttt{CIP}}, \mathcal{F}^{\mathcal{L}}_{\texttt{ZK,KS,IP}}\}, \mathcal{S}_{\mathcal{A}})$ ***if*** $\texttt{CIP}_{\texttt{DLP}}$ *and* $\texttt{CIP}_{\texttt{SIS}}$ *are complete, DLP is hard for* $(\mathbb{G}, q)$*, and Approx-SIS is hard for* $(n, m+1, q, \gamma, \gamma', N)$*, respectively.*

Completeness. All of our proofs try to show the knowledge of some multi-variable equations. For multiplication, the equation is $(a_1 + x_2v_1)(a_2 + x_2v_2) + x_0(a_1 + x_2v_1) + x_1(a_2 + x_2v_2)$ which is $(x_2)^2v_1v_2 + x_2(v_1a_2 + v_2a_1 + x_0v_1 + x_1v_2) + x_0a_1 + x_1a_2 + a_1a_2)$. Since v_1v_2 should be equal to v_0, the equation can be written as, $(x_2)^2v_0 + x_2(v_1a_2 + v_2a_1 + x_0v_1 + x_1v_2) + x_0a_1 + x_1a_2 + a_1a_2)$. Also, CIP-SIS's range proofs try to show the following equation for some bit b that should be either 1 or 0; $(a_1 + x_2b)((a_1 + x_2b) - x2) \overset{?}{=} x_2a_1(2b-1) + a_1^2$. The equality checks a simple equation, $v_1 - v_2 \overset{?}{=} 0$. The division of $v_1 = v_0v_2 + v_3$ is similar to the multiplication which is $(a_0 + x_2v_0)(a_2 + x_2v_2) + x_0(a_0 + x_2v_0) + x_1(a_2 + x_2v_2)$. Once the equation is expanded with the remainder v_3, it is $(x_2)^2(v_1 - v_3) + x_2(v_0a_2 + v_2a_0 + x_0v_0 + x_1v_2) + x_0a_0 + x_1a_2 + a_0a_2)$. Therefore, we claim that our protocols are complete. □

Hiding and Binding of Confidential Integers. Theorem 5 proves that confidential integers are hiding and binding when DLP is hard for $(\mathbb{G}, q)$, and Approx-SIS is hard for $(n, m+1, q, \gamma, \gamma', N)$. □

Zero-Knowledge and Intra-Privacy. Theorem 6 proves zero-knowledge of CIP-DLP's range proofs. We prove zero-knowledge and intra-privacy by contradiction for the rest of primitive relations. Let $\texttt{CIP}_{\texttt{DLP}}$ and $\texttt{CIP}_{\texttt{SIS}}$ not be zero-knowledge and intra-private. Each proof and transcript has two types of components in $(\mathbb{Z}_q^+, \mathbb{G})$ for CIP-DLP and $(\mathbb{X}^*, \mathbb{X}_q^n)$ for CIP-SIS. Here, all $\mathbb{Z}_q^+$ and $\mathbb{X}^*$ are statistically hiding due to modular operations and rejection sampling in CIP-DLP and CIP-SIS. Therefore, if an adversary breaks zero-knowledge and intra-privacy of CIP-DLP and CIP-SIS, then the adversary distinguishes properly generated $\mathbb{X}_q^n$ and $\mathbb{G}$ over randomly simulated components; meaning that the adversary breaks the hiding property of DLP and Approx-SIS. Hence, $\texttt{CIP}_{\texttt{DLP}}$ and $\texttt{CIP}_{\texttt{SIS}}$ are zero-knowledge and intra-private if DLP and Approx-SIS is hard. □

Knowledge Soundness. Recall that our proofs check some kind of equation with hash challenges, e.g., the "multiplication" proof checks that; RHS's exponent is $(x_2)^2v_0 + x_2(v_1a_2 + v_2a_1 + x_0v_1 + x_1v_2) + x_0a_1 + x_1a_2 + a_1a_2)$ and LHS's exponent is $(a_1 + x_2v_1)(a_2 + x_2v_2) + x_0(a_1 + x_2v_1) + x_1(a_2 + x_2v_2)$. If a cheating verifier computes LHS and RHS to be equal according to the equation (see confidential multiplication of $\mathtt{CIP}_{\mathtt{DLP}}$ and $\mathtt{CIP}_{\mathtt{SIS}}$) but $v_0 \neq v_1 v_2$ then the cheating prover breaks the biding property of DLP or SIS since they find a commitment with two different openings. Similarly, if cheating verifier creates "cheating but valid" proofs for any primitive relation in $\mathcal{L}$ then this cheating verifier can break the binding property of $\mathtt{CIP}_{\mathtt{DLP}}$ and $\mathtt{CIP}_{\mathtt{SIS}}$. Since our composed relations are sequential proofs of primitive relations, "cheating but valid" proofs are created by breaking the binding property once or multiple times. Therefore, we claim $\mathtt{CIP}_{\mathtt{DLP}}$ and $\mathtt{CIP}_{\mathtt{SIS}}$ are knowledge sound (more specifically simulation extractable) if DLP and Approx-SIS is hard (see Theorem 5). □

Finally, we claim that Theorem 7 is valid since $\mathtt{CIP}_{\mathtt{DLP}}$ and $\mathtt{CIP}_{\mathtt{SIS}}$ are complete, binding, zero-knowledge, simulation extractable, and intra-private if DLP and Approx-SIS are hard, respectively.

With Theorem 4 and Theorem 7, we have proven that Theorem 3 is valid.

Conclusion Confidential Integer Processing (CIP) enables computations of hidden integers while proving that the computations were performed correctly. We propose two CIP protocols based on the Discrete Logarithmic Problem (DLP) and Approximate Modular Ring Shortest Integer Solution Problem (Approx-SIS). Our CIP protocols are zero-knowledge — the verifiers learn nothing except the program's path for the given inputs — and intra-private — participants can do multi-party computations without sharing all of their hidden integers with others. Also, our protocols are proven to be Universally Secure according to the Universal Composability (UC) framework, and our UC model can be used as an exemplary model for complex zero-knowledge protocols. Our CIP protocols have a wide range of applications, e.g., (1) zero-knowledge smart contracts for Ring Confidential Transactions and Aggregable Confidential Transactions, and (2) private multi-party computations for federated machine learning, outsourced-security software, and multi-party cloud computing.

References

1. Alberto Torres, W., Kuchta, V., Steinfeld, R., Sakzad, A., Liu, J.K., Cheng, J.: Lattice ringct v2.0 with multiple input and multiple output wallets. In: Jang-Jaccard, J., Guo, F. (eds.) Information Security and Privacy. pp. 156–175. Springer International Publishing, Cham (2019)
2. Alupotha, J., Boyen, X., Mckague, M.: Aggregable confidential transactions for efficient quantum-safe cryptocurrencies. IEEE Access **10**, 17722–17747 (2022)
3. Ames, S., Hazay, C., Ishai, Y., Venkitasubramaniam, M.: Ligero: Lightweight sublinear arguments without a trusted setup. In: Proceedings of the 2017 acm sigsac conference on computer and communications security. pp. 2087–2104 (2017)
4. Ben-Sasson, E., Bentov, I., Horesh, Y., Riabzev, M.: Scalable zero knowledge with no trusted setup. In: Annual international cryptology conference. pp. 701–732. Springer (2019)
5. Ben-Sasson, E., Chiesa, A., Riabzev, M., Spooner, N., Virza, M., Ward, N.P.: Aurora: Transparent succinct arguments for r1cs. In: Annual international conference on the theory and applications of cryptographic techniques. pp. 103–128. Springer (2019)

6. Blum, M.: How to prove a theorem so no one else can claim it. In: Proceedings of the International Congress of Mathematicians. vol. 1, p. 2. Citeseer (1986)
7. Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečnỳ, J., Mazzocchi, S., McMahan, B., et al.: Towards federated learning at scale: System design. Proceedings of Machine Learning and Systems **1**, 374–388 (2019)
8. Boschini, C., Camenisch, J., Ovsiankin, M., Spooner, N.: Efficient post-quantum snarks for rsis and rlwe and their applications to privacy. PQCrypto **12100**, 247–267 (2020)
9. Bowe, S., Chiesa, A., Green, M., Miers, I., Mishra, P., Wu, H.: Zexe: Enabling decentralized private computation. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 947–964. IEEE (2020)
10. Brakerski, Z., Vaikuntanathan, V.: Efficient fully homomorphic encryption from (standard) lwe. SIAM Journal on Computing **43**(2), 831–871 (2014)
11. Brickell, E., Pointcheval, D., Vaudenay, S., Yung, M.: Design validations for discrete logarithm based signature schemes. In: International Workshop on Public Key Cryptography. pp. 276–292. Springer (2000)
12. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world
13. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: International Conference on Financial Cryptography and Data Security. pp. 423–443. Springer (2020)
14. Bünz, B., Bootle, J., Boneh, D., Poelstra, A., Wuille, P., Maxwell, G.: Bulletproofs: Efficient range proofs for confidential transactions. IEEE SP citation_publication_date= May 2018 (2017)
15. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: Proceedings 42nd IEEE Symposium on Foundations of Computer Science. pp. 136–145. IEEE (2001)
16. Canetti, R.: Universally composable signature, certification, and authentication. In: Proceedings. 17th IEEE Computer Security Foundations Workshop, 2004. pp. 219–233. IEEE (2004)
17. Canetti, R., Fischlin, M.: Universally composable commitments. In: Annual International Cryptology Conference. pp. 19–40. Springer (2001)
18. Chen, Y., Genise, N., Mukherjee, P.: Approximate trapdoors for lattices and smaller hash-and-sign signatures. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 3–32. Springer (2019)
19. Cheng, R., Zhang, F., Kos, J., He, W., Hynes, N., Johnson, N., Juels, A., Miller, A., Song, D.: Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contracts. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 185–200. IEEE (2019)
20. Chiesa, A., Ojha, D., Spooner, N.: Fractal: Post-quantum and transparent recursive proofs from holography. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 769–793. Springer (2020)
21. Damgård, I., Nielsen, J.B.: Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In: Annual International Cryptology Conference. pp. 581–596. Springer (2002)
22. Dolev, S., Wang, Z.: Sodsmpc: Fsm based anonymous and private quantum-safe smart contracts. In: 2020 IEEE 19th International Symposium on Network Computing and Applications (NCA). pp. 1–10. IEEE (2020)

23. Eberhardt, J., Tai, S.: Zokrates-scalable privacy-preserving off-chain computations. In: 2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData). pp. 1084–1091. IEEE (2018)
24. Esgin, M.F., Steinfeld, R., Zhao, R.K.: Matrict+: More efficient post-quantum private blockchain payments. Cryptology ePrint Archive (2021)
25. Esgin, M.F., Zhao, R.K., Steinfeld, R., Liu, J.K., Liu, D.: Matrict: efficient, scalable and post-quantum blockchain confidential transactions protocol. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 567–584 (2019)
26. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Conference on the theory and application of cryptographic techniques. pp. 186–194. Springer (1986)
27. Foundation, E.: The solidity contract-oriented programming language ((accessed on 2022-01-24)), https://github.com/ethereum/solidity
28. Fuchsbauer, G., Orrù, M., Seurin, Y.: Aggregate cash systems: A cryptographic investigation of mimblewimble. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 657–689. Springer (2019)
29. Gennaro, R., Minelli, M., Nitulescu, A., Orrù, M.: Lattice-based zk-snarks from square span programs. In: Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security. pp. 556–573 (2018)
30. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the forty-first annual ACM symposium on Theory of computing. pp. 169–178 (2009)
31. Gentry, C., Wichs, D.: Separating succinct non-interactive arguments from all falsifiable assumptions. In: Proceedings of the forty-third annual ACM symposium on Theory of computing. pp. 99–108 (2011)
32. Hofheinz, D., Müller-Quade, J., Unruh, D.: Universally composable zero-knowledge arguments and commitments from signature cards. In: 5th Central European Conference on Cryptology (2005)
33. https://www.beam.mw/: Scalable confidential cryptocurrency - mimblewimble implementation ((accessed on 2021-01-27)), https://www.beam.mw/
34. Ishai, Y., Su, H., Wu, D.J.: Shorter and faster post-quantum designated-verifier zksnarks from lattices. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 212–234 (2021)
35. Jedusor, T.E.: Mimblewimble (2016), https://docs.beam.mw/Mimblewimble.pdf
36. Kairouz, P., McMahan, H.B., Avent, B., Bellet, A., Bennis, M., Bhagoji, A.N., Bonawitz, K., Charles, Z., Cormode, G., Cummings, R., et al.: Advances and open problems in federated learning. Foundations and Trends® in Machine Learning **14**(1–2), 1–210 (2021)
37. Kalodner, H., Goldfeder, S., Chen, X., Weinberg, S.M., Felten, E.W.: Arbitrum: Scalable, private smart contracts. In: 27th USENIX Security Symposium (USENIX Security 18). pp. 1353–1370. USENIX Association, Baltimore, MD (Aug 2018), https://www.usenix.org/conference/usenixsecurity18/presentation/kalodner
38. Kilian, J.: A note on efficient zero-knowledge proofs and arguments. In: Proceedings of the twenty-fourth annual ACM symposium on Theory of computing. pp. 723–732 (1992)

39. Kosba, A., Miller, A., Shi, E., Wen, Z., Papamanthou, C.: Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In: 2016 IEEE symposium on security and privacy (SP). pp. 839–858. IEEE (2016)
40. Lyubashevsky, V.: Fiat-shamir with aborts: Applications to lattice and factoring-based signatures. In: International Conference on the Theory and Application of Cryptology and Information Security. pp. 598–616. Springer (2009)
41. McMahan, B., Ramage, D.: Federated learning: Collaborative machine learning without centralized training data ((accessed on 2021-12-05)), `https://ai.googleblog.com/2017/04/federated-learning-collaborative.html`
42. Micali, S.: Cs proofs. In: Proceedings 35th Annual Symposium on Foundations of Computer Science. pp. 436–453. IEEE (1994)
43. Mouris, D., Tsoutsos, N.G.: Zilch: A framework for deploying transparent zero-knowledge proofs. IEEE Transactions on Information Forensics and Security (2021)
44. Nitulescu, A.: Lattice-based zero-knowledge snargs for arithmetic circuits. In: International Conference on Cryptology and Information Security in Latin America. pp. 217–236. Springer (2019)
45. Noether, S., Noether, S.: Monero is not that mysterious. Technical report (2014), online available at: https://web.getmonero.org/ru/resources/research-lab/pubs/MRL-0003.pdf
46. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual International Cryptology Conference. pp. 129–140. Springer (1991)
47. Poelstra, A.: Scriptless scripts (accessed on 2022-4-21), `https://download.wpsoftware.net/bitcoin/wizardry/mw-slides/2017-03-mit-bitcoin-expo/slides.pdf`
48. PQ-Crystals: Dilithium signature scheme (2019), `https://github.com/pq-crystals/dilithium`
49. Schnorr, C.P.: Efficient identification and signatures for smart cards. In: Conference on the Theory and Application of Cryptology. pp. 239–252. Springer (1989)
50. Steffen, S., Bichsel, B., Baumgartner, R., Vechev, M.: Zeestar: Private smart contracts by homomorphic encryption and zero-knowledge proofs. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 1543–1543. IEEE Computer Society (2022)
51. Steffen, S., Bichsel, B., Gersbach, M., Melchior, N., Tsankov, P., Vechev, M.: zkay: Specifying and enforcing data privacy in smart contracts. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 1759–1776 (2019)
52. Sun, S.F., Au, M.H., Liu, J.K., Yuen, T.H.: Ringct 2.0: A compact accumulator-based (linkable ring signature) protocol for blockchain cryptocurrency monero. In: European Symposium on Research in Computer Security. pp. 456–474. Springer (2017)
53. grin tech.org: Minimal implementation of the mimblewimble protocol ((accessed on 2021-01-27)), `https://github.com/mimblewimble/grin`
54. Van Dijk, M., Gentry, C., Halevi, S., Vaikuntanathan, V.: Fully homomorphic encryption over the integers. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 24–43. Springer (2010)
55. Whitworth, M.: Outsourced security–the benefits and risks. Network Security **2005**(10), 16–19 (2005)
56. Wood, G.: Ethereum: A secure decentralised generalised transaction ledger. Ethereum project yellow paper **151**, 1–32 (2014)

57. Zhang, H., Zhang, F., Wei, B., Du, Y.: Implementing confidential transactions with lattice techniques. IET Information Security **14**(1), 30–38 (2019)
58. Zyskind, G., Nathan, O., Pentland, A.: Enigma: Decentralized computation platform with guaranteed privacy. arXiv preprint arXiv:1506.03471 (2015)

A Division

```
$\triangleright$ Proving Confidential Division
$\underline{\texttt{CIP}_{\texttt{DLP}}.\texttt{prove}((\texttt{u})\texttt{div}; \pi_0{=}\{c_0, c_1, c_2, c_3\}}$
        $\underline{\boldsymbol{k} = \{v_0, k_0, v_1, k_1, v_2, k_2, v_3, k_3\}):}$
$e = h^1 \in \mathbb{G}$
if $v_0 \overset{?}{=} v_1 // v_2 \vee v_3 \overset{?}{=} v_1 - v_0 v_2 \pmod{q}$:
        return $\perp$
$k'_1 \xleftarrow{\$} \mathbb{Z}_q^+ \; k'_2 \xleftarrow{\$} \mathbb{Z}_q^+$
if "div" $\triangleright$ signed numbers
    if $v_0, v_1, v_2, v_3 \notin (-2^L, 2^L)$: return $\perp$
    $(k'_2, c'_2) = \texttt{CIP}_{\texttt{DLP}}.\texttt{cint}((v_2)^2)$
    $(k'_3, c'_3) = \texttt{CIP}_{\texttt{DLP}}.\texttt{cint}((v_3 - 1)^2)$
    $\pi'_1{=}\texttt{CIP}_{\texttt{DLP}}.\texttt{prove}\big(\texttt{mul}; \pi_0{=}\{c'_2, c_2, c_2\},$
        $\boldsymbol{k} = \{(v_2)^2, k'_2, v_2, k_2, v_2, k_2\}\big)$
    $\pi'_2{=}\texttt{CIP}_{\texttt{DLP}}.\texttt{prove}\big(\texttt{mul};$
        $\pi_0{=}\{c'_3, c_3 e^{-1}, c_3 e^{-1}\},$
        $\boldsymbol{k}{=}\{(v_3{-}1)^2, k'_3, v_3{-}1, k_3, v_3{-}1, k_3\}\big)$
    $\pi'_3{=}\texttt{CIP}_{\texttt{SIS}}.\texttt{prove}\big(\texttt{leq}; \pi_0{=}\{c'_3, c'_2\},$
        $\boldsymbol{k}{=}\{(v_3{-}1)^2, k'_3, (v_2)^2, k'_2\}\big)$
            $\triangleright$ leq proof is for $(2L{+}1)$-bits
    $\pi'_4 = \texttt{CIP}_{\texttt{SIS}}.\texttt{prove}\big(\texttt{range}_L;$
        $\pi_0{=}\{c_3\}, \boldsymbol{k} = \{(v_3, k_3\}\big)$
    $\pi' = (c'_2, c'_3, \pi'_1, \pi'_2, \pi'_3, \pi'_4)$
if "udiv" $\triangleright$ unsigned numbers
    if $v_0, v_1, v_2, v_3 \notin [0, 2^L)$: return $\perp$
    $\pi'{=}\texttt{CIP}_{\texttt{SIS}}.\texttt{prove}\big(\texttt{leq}; \pi_0{=}\{c_3 e^{-1}, c_2\},$
        $\boldsymbol{k} = \{(v_3{-}1), k_3, v_2, k_2\}\big)$
$a_0 \xleftarrow{\$} \mathbb{Z}_q^+; a_2 \xleftarrow{\$} \mathbb{Z}_q^+$ $\triangleright$ Masks
$t_1 = g^{k'_1} h^{v_0 a_2 + v_2 a_0} \in \mathbb{G}$
$\triangleright$ Challenge 0
$x_0 = \mathsf{hash}(c_0, c_1, c_2, c_3, t_1) \in \mathbb{Z}_q^+$
$x_1 = \mathsf{hash}(x_0) \in \mathbb{Z}_q^+$ $\triangleright$ Challenge 1
$t_2 = g^{k'_2} h^{a_0 a_2 + x_0 a_0 + x_1 a_2} \in \mathbb{G}$
$x_2 = \mathsf{hash}(x_1, t_2) \in \mathbb{Z}_q^+$ $\triangleright$ Challenge 2
$z_0 = v_0 x_2 + a_0 \in \mathbb{X}$; $z_2 = v_2 x_2 + a_2 \in \mathbb{X}$
$s = x_2^2(k_1{-}k_3){+}x_2(k'_1{+}x_0 k_0{+}x_1 k_2){+}k'_2$
return $\pi = (z_0, z_2, s, t_1, t_2, \pi')$
```

```
$\triangleright$ Confidential Division Verification
$\underline{\texttt{CIP}_{\texttt{DLP}}.\texttt{ver}((\texttt{u})\texttt{div}; \pi_0{=}\{c_0, c_1, c_2, c_3\}, \pi)}$
$(z_0, z_2, s, t_1, t_2, \pi') := \pi$
$e = h^1 \in \mathbb{G}$
$x_0 = \mathsf{hash}(c_0, c_1, c_2, \vec{\boldsymbol{t}}_1) \in \mathbb{Z}_q^+$
$x_1 = \mathsf{hash}(x_0) \in \mathbb{Z}_q^+$
$\vec{s}_1 = (z_0 z_2 + x_0 z_0 + x_1 z_2) \in \mathbb{Z}_q^+$
if "div" $\triangleright$ signed numbers
        $(c'_2, c'_3, \pi'_1, \pi'_2, \pi'_3, \pi'_4) := \pi'$
        if $\neg\texttt{CIP}_{\texttt{SIS}}.\texttt{ver}(\texttt{mul}; \{c'_2, c_2, c_2\}, \pi'_1)$
        $\vee\neg\texttt{CIP}_{\texttt{SIS}}.\texttt{ver}\big(\texttt{mul};$
                $\{c'_3, c_3 e^{-1}, c_3 e^{-1}\}, \pi'_2)$
        $\vee\neg\texttt{CIP}_{\texttt{SIS}}.\texttt{prove}\big(\texttt{leq}; \{c'_3, c'_2\}, \pi'_3\big)$
        $\vee\neg\texttt{CIP}_{\texttt{SIS}}.\texttt{prove}\big(\texttt{range}_L; \{c_3\}, \pi'_4\big)$:
                return 0
if "udiv" $\triangleright$ unsigned numbers
    if $\neg\texttt{CIP}_{\texttt{SIS}}.\texttt{prove}\big(\texttt{leq}; \pi_0{=}\{c_3 e^{-1},$
                $c_2\}, \pi)$: return 0
return $g^s h^{z_0 z_2 + x_0 z_0 + x_1 z_2} \overset{?}{=} (c_1 c_3^{-1})^{(x_2)^2}$
        $+(t_1 \times c_0^{x_0} \times c_2^{x_1})^{x_2} \times t_2 \in \mathbb{G}$
```

```
$\triangleright$ Proving Confidential Division
$\underline{\texttt{CIP}_{\texttt{SIS}}.\texttt{prove}((\texttt{u})\texttt{div}; \pi_0{=}\{\vec{\boldsymbol{c}}_0, \vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2, \vec{\boldsymbol{c}}_3\}}$
        $\underline{\boldsymbol{k} = \{v_0, \vec{\boldsymbol{k}}_0, v_1, \vec{\boldsymbol{k}}_1, v_2, \vec{\boldsymbol{k}}_2, v_3, \vec{\boldsymbol{k}}_3\}):}$
$\vec{\boldsymbol{e}} = \vec{\boldsymbol{V}}\mathsf{rot}(1, 0) \in \mathbb{X}_q^n$
if $v_0 \overset{?}{=} v_1 // v_2 \vee v_3 \overset{?}{=} v_1 - v_0 v_2$: return $\perp$
$\vec{\boldsymbol{k}}'_1 \xleftarrow{\$} [-\tau_1, \tau_1]^{m \times N}$
$\vec{\boldsymbol{k}}'_2 \xleftarrow{\$} [-\tau_2, \tau_2]^{m \times N}$
if "div" $\triangleright$ signed numbers
    if $v_0, v_1, v_2, v_3 \notin (-2^L, 2^L)$: return $\perp$
    $(\vec{\boldsymbol{k}}'_2, \vec{\boldsymbol{c}}'_2) = \texttt{CIP}_{\texttt{SIS}}.\texttt{cint}((v_2)^2)$
    $(\vec{\boldsymbol{k}}'_3, \vec{\boldsymbol{c}}'_3) = \texttt{CIP}_{\texttt{SIS}}.\texttt{cint}((v_3 - 1)^2)$
    $\pi'_1{=}\texttt{CIP}_{\texttt{SIS}}.\texttt{prove}\big(\texttt{mul}; \pi_0{=}\{\vec{\boldsymbol{c}}'_2, \vec{\boldsymbol{c}}_2, \vec{\boldsymbol{c}}_2\},$
        $\boldsymbol{k} = \{(v_2)^2, \vec{\boldsymbol{k}}'_2, v_2, \vec{\boldsymbol{k}}_2, v_2, \vec{\boldsymbol{k}}_2\}\big)$
    $\pi'_2{=}\texttt{CIP}_{\texttt{SIS}}.\texttt{prove}\big(\texttt{mul};$
        $\pi_0{=}\{\vec{\boldsymbol{c}}'_3, \vec{\boldsymbol{c}}_3 - \vec{\boldsymbol{e}}, \vec{\boldsymbol{c}}_3 - \vec{\boldsymbol{e}}\},$
        $\boldsymbol{k}{=}\{(v_3{-}1)^2, \vec{\boldsymbol{k}}'_3,$
            $v_3{-}1, \vec{\boldsymbol{k}}_3, v_3{-}1, \vec{\boldsymbol{k}}_3\}\big)$
    $\pi'_3{=}\texttt{CIP}_{\texttt{SIS}}.\texttt{prove}\big(\texttt{leq}; \pi_0{=}\{\vec{\boldsymbol{c}}'_3, \vec{\boldsymbol{c}}'_2\},$
19:         $\boldsymbol{k} = \{(v_3-1)^2, \vec{\boldsymbol{k}}'_3, (v_2)^2, \vec{\boldsymbol{k}}'_2\}\big)$
20:             ▷ leq proof is for $(2L+1)$-bits
21:     $\pi'_4 = \texttt{CIP}_{\texttt{SIS}}.\texttt{prove}\big(\texttt{range}_L;$
22:         $\pi_0=\{\vec{\boldsymbol{c}}_3\}$, $\boldsymbol{k} = \{(v_3, \vec{\boldsymbol{k}}_3\}\big)$
23:     $\pi' = (\vec{\boldsymbol{c}}'_2, \vec{\boldsymbol{c}}'_3, \pi'_1, \pi'_2, \pi'_3, \pi'_4)$
24: if "udiv" ▷ unsigned numbers
25:     if $v_0, v_1, v_2, v_3 \notin [0, 2^L)$: return $\perp$
26:     $\pi'=\texttt{CIP}_{\texttt{SIS}}.\texttt{prove}\big(\texttt{leq}; \pi_0=\{\vec{\boldsymbol{c}}_3-\vec{\boldsymbol{e}}, \vec{\boldsymbol{c}}_2\},$
27:         $\boldsymbol{k} = \{(v_3-1), \vec{\boldsymbol{k}}_3, v_2, \vec{\boldsymbol{k}}_2\}\big)$
28: ▷ Lazy Sampling when $w = 2^L - 1$
29: $\vec{a}_0 \xleftarrow{\$} [-(\alpha_1-w+v_0), \alpha_1-w+v_0]^N \in \mathbb{X}$
30: $\vec{a}_2 \xleftarrow{\$} [-(\alpha_1-w+v_2), \alpha_1-w+v_2]^N \in \mathbb{X}$
31: $\vec{\boldsymbol{t}}_1 = \texttt{highbits}_{p_1}(\vec{\boldsymbol{V}}(v_0\vec{a}_2 + v_2\vec{a}_0)$
32:         $+\vec{\boldsymbol{K}}\vec{\boldsymbol{k}}'_1) \in \mathbb{X}_q^n$
33: ▷ Challenge 0
34: $\vec{x}_0 = \mathsf{hash}(\vec{\boldsymbol{c}}_0, \vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2, \vec{\boldsymbol{c}}_3, \vec{\boldsymbol{t}}_1) \in \mathcal{C}_\beta^N$
35: ▷ Challenge 1
36: $\vec{x}_1 = \mathsf{hash}(\vec{x}_0) \in \mathcal{C}_\beta^N$
37: $\vec{a}' = \vec{a}_0\vec{a}_2 + \vec{x}_0\vec{a}_0 + \vec{x}_1\vec{a}_2 \in \mathbb{X}$
38: $\vec{\boldsymbol{t}}_2 = \texttt{highbits}_{\gamma'}(\vec{\boldsymbol{V}}\vec{a}' + \vec{\boldsymbol{K}}\vec{\boldsymbol{k}}'_2) \in \mathbb{X}_q^n$
39: ▷ Challenge 2
40: $\vec{x}_2 = \mathsf{hash}(\vec{x}_1, \vec{\boldsymbol{t}}_2) \in \mathcal{C}_\beta^N$
41: $\vec{z}_0 = v_0\vec{x}_2 + \vec{a}_0 \in \mathbb{X}$
42: $\vec{z}_2 = v_2\vec{x}_2 + \vec{a}_2 \in \mathbb{X}$
43: $\vec{\boldsymbol{s}} = \vec{x}_2^2(\vec{\boldsymbol{k}}_1-\vec{\boldsymbol{k}}_3)+\vec{x}_2(\vec{\boldsymbol{k}}'_1+\vec{x}_0\vec{\boldsymbol{k}}_0+\vec{x}_1\vec{\boldsymbol{k}}_2)$
44:             $+\vec{\boldsymbol{k}}'_2 \in \mathbb{X}^m$
45: $\vec{s}_1 = (\vec{z}_0\vec{z}_2 + \vec{x}_0\vec{z}_0 + \vec{x}_1\vec{z}_2) \in \mathbb{X}$
46: if $\|\vec{z}_0\|>(\alpha-w) \vee \|\vec{z}_2\|>(\alpha - w)\vee$
47:     $\|\vec{s}_1\|>\gamma \vee \|\vec{\boldsymbol{s}}\|>(\tau_2 - 4\beta^2\tau - \beta\tau_1)$ :
48:         go to Step 6
49: ▷ Hints for $\vec{\boldsymbol{t}}_2$
50: $\vec{\boldsymbol{t}}'_2=\texttt{highbits}_{\gamma'}(\vec{\boldsymbol{K}}\vec{\boldsymbol{s}}+\vec{\boldsymbol{V}}\vec{\boldsymbol{s}}_1-\vec{x}_2^2(\vec{\boldsymbol{c}}_1-\vec{\boldsymbol{c}}_3)$
51:     $-\vec{x}_2(\texttt{up}_{p_1}(\vec{\boldsymbol{t}}_1) + \vec{x}_0\vec{\boldsymbol{c}}_0 + \vec{x}_1\vec{\boldsymbol{c}}_2)) \in \mathbb{X}_q^n$
52: $\vec{\boldsymbol{h}} = \texttt{hints}(\vec{\boldsymbol{t}}'_2, \vec{\boldsymbol{t}}_2)$
53: if $\vec{\boldsymbol{h}} =\perp$: go to Step 6
54: return $\pi = (\vec{z}_0, \vec{z}_2, \vec{\boldsymbol{s}}, \vec{\boldsymbol{t}}_1, \vec{x}_2, \vec{\boldsymbol{h}}, \pi')$
```

```
 1: ▷ Confidential Division Verification
 2: $\texttt{CIP}_{\texttt{SIS}}.\texttt{ver}((\texttt{u})\texttt{div}; \pi_0=\{\vec{\boldsymbol{c}}_0, \vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2, \vec{\boldsymbol{c}}_3\}, \pi)$
 3: $(\vec{z}_0, \vec{z}_2, \vec{\boldsymbol{s}}, \vec{\boldsymbol{t}}_1, \vec{x}_2, \vec{\boldsymbol{h}}, \pi') := \pi$
 4: $\vec{\boldsymbol{e}} = \vec{\boldsymbol{V}}\mathsf{rot}(1, 0) \in \mathbb{X}_q^n$
 5: $\vec{x}_0 = \mathsf{hash}(\vec{\boldsymbol{c}}_0, \vec{\boldsymbol{c}}_1, \vec{\boldsymbol{c}}_2, \vec{\boldsymbol{t}}_1) \in \mathcal{C}_\beta^N$
 6: $\vec{x}_1 = \mathsf{hash}(\vec{x}_0) \in \mathcal{C}_\beta^N$
 7: $\vec{s}_1 = (\vec{z}_0\vec{z}_2 + \vec{x}_0\vec{z}_0 + \vec{x}_1\vec{z}_2) \in \mathbb{X}$
 8: if $\|[\vec{z}_0, \vec{z}_2]\| > \alpha \vee \|\vec{\boldsymbol{s}}\| > \tau_2 \vee \|\vec{s}_1\| > \gamma$:
 9:         return 0
10: if "div" ▷ signed numbers
11:         $(\vec{\boldsymbol{c}}'_2, \vec{\boldsymbol{c}}'_3, \pi'_1, \pi'_2, \pi'_3, \pi'_4) := \pi'$
12:         if $\neg\texttt{CIP}_{\texttt{SIS}}.\texttt{ver}(\texttt{mul}; \{\vec{\boldsymbol{c}}'_2, \vec{\boldsymbol{c}}_2, \vec{\boldsymbol{c}}_2\}, \pi'_1)$
13:         $\vee\neg\texttt{CIP}_{\texttt{SIS}}.\texttt{ver}\big(\texttt{mul};$
14:                 $\{\vec{\boldsymbol{c}}'_3, \vec{\boldsymbol{c}}_3 - \vec{\boldsymbol{e}}, \vec{\boldsymbol{c}}_3 - \vec{\boldsymbol{e}}\}, \pi'_2)$
15:         $\vee\neg\texttt{CIP}_{\texttt{SIS}}.\texttt{prove}\big(\texttt{leq}; \{\vec{\boldsymbol{c}}'_3, \vec{\boldsymbol{c}}'_2\}, \pi'_3\big)$
16:         $\vee\neg\texttt{CIP}_{\texttt{SIS}}.\texttt{prove}\big(\texttt{range}_L; \{\vec{\boldsymbol{c}}_3\}, \pi'_4\big)$:
17:                 return 0
18: if "udiv" ▷ unsigned numbers
19:     if $\neg\texttt{CIP}_{\texttt{SIS}}.\texttt{prove}\big(\texttt{leq}; \pi_0=\{\vec{\boldsymbol{c}}_3-\vec{\boldsymbol{e}},$
20:                 $\vec{\boldsymbol{c}}_2\}, \pi)$ : return 0
21: ▷ Recreate $\vec{\boldsymbol{t}}_2$
22: $\vec{\boldsymbol{t}}'_2=\texttt{highbits}_{\gamma'}(\vec{\boldsymbol{K}}\vec{\boldsymbol{s}}+\vec{\boldsymbol{V}}\vec{\boldsymbol{s}}_1-\vec{x}_2^2(\vec{\boldsymbol{c}}_1-\vec{\boldsymbol{c}}_3)$
23:     $-\vec{x}_2(\texttt{up}_{p_1}(\vec{\boldsymbol{t}}_1) + \vec{x}_0\vec{\boldsymbol{c}}_0 + \vec{x}_1\vec{\boldsymbol{c}}_2)) \in \mathbb{X}_q^n$
24: $\vec{\boldsymbol{t}}_2 = \texttt{use\_hints}(\vec{\boldsymbol{t}}'_2, \vec{\boldsymbol{h}})$
25: return $\vec{\boldsymbol{t}} \neq\perp \wedge\vec{x}_2 \stackrel{?}{=} \mathsf{hash}(\vec{x}_1, \vec{\boldsymbol{t}}_2)$
```

B Binary Commitments to Confidential Integers

This section explains how to verify whether a binary lattice commitment holds the same integer as a confidential integer or not. For iMSIS $(n', m'+L, q', B, N)$, let a binary commitment be $\vec{\boldsymbol{b}} = \vec{\boldsymbol{V}}'[\boldsymbol{b}] + \vec{\boldsymbol{K}}'\vec{\boldsymbol{k}}' \in \mathbb{X}_{q'}^{n'}$ when $[\vec{\boldsymbol{V}}', \vec{\boldsymbol{K}}']$ is a random matrix and $v = \sum_{i=0}^{L-1} b_i$. Then the proof work as follows when $\tau' \ll \tau'_3 \leq B$ and $LN\alpha \leq B$.

- Pick random $\vec{\boldsymbol{y}} \in [\tau'_3, \tau'_3]^{m\times N}$ and $\vec{\boldsymbol{y}}' \in [\tau', \tau']^{m'\times N'}$.
- For all $\boldsymbol{b}$ get random $\vec{\boldsymbol{a}} \in [-(\alpha - (1 - b_i)), \alpha - (1 - b_i)]^{L\times N}$.
- Get $\vec{\boldsymbol{t}} = \texttt{hightbits}_{p_1}(\vec{\boldsymbol{V}} \sum_{i=0}^{L-1} \vec{a}_i + \vec{\boldsymbol{K}}\vec{\boldsymbol{y}}) \in \mathbb{X}_q^n$ and $\vec{\boldsymbol{t}}' = \vec{\boldsymbol{V}}'\vec{\boldsymbol{a}} + \vec{\boldsymbol{K}}'\vec{\boldsymbol{y}} \in \mathbb{X}_{q'}^{n'}$
- Compute $\vec{x} = \texttt{hash}(\vec{\boldsymbol{c}}, \vec{\boldsymbol{b}}, \vec{\boldsymbol{t}}, \vec{\boldsymbol{t}}') \in \mathcal{C}_\beta^N$.
- Set $\vec{\boldsymbol{z}} = [\vec{a}_i+\texttt{rot}(i, b_i)\vec{x}]_{i=0}^{L-1} \in \mathbb{X}^L$, $\vec{\boldsymbol{s}} = \vec{\boldsymbol{y}}+\vec{x}\vec{\boldsymbol{k}} \in \mathbb{X}^m$, and $\vec{\boldsymbol{s}}' = \vec{\boldsymbol{y}}'+\vec{x}\vec{\boldsymbol{k}}' \in \mathbb{X}^{m'}$.
- Perform rejection sampling if $\|\vec{\boldsymbol{z}}\| > (\alpha-1)$, $\|\vec{\boldsymbol{s}}\| > (\tau_3-\tau)$ or $\|\vec{\boldsymbol{s}}'\| > (\tau'_3-\tau')$. If samples are rejected, go to the first step.

- Compute hints $\vec{\boldsymbol{h}} = \texttt{hints}(\vec{\boldsymbol{V}} \sum_{i=0}^{L-1} \vec{z}_i + \vec{\boldsymbol{K}}\vec{\boldsymbol{s}} - \vec{\boldsymbol{c}}, \vec{\boldsymbol{t}})$.
- The proof is $(\vec{x}, \vec{\boldsymbol{z}}, \vec{\boldsymbol{s}}, \vec{\boldsymbol{s}}', \vec{\boldsymbol{h}})$.

The verification works as follows.

- Recompute $\vec{\boldsymbol{t}} = \texttt{use\_hints}(\texttt{hightbits}_{p_1}(\vec{\boldsymbol{V}} \sum_{i=0}^{L-1} \vec{z}_i + \vec{\boldsymbol{K}}\vec{\boldsymbol{s}} - \vec{\boldsymbol{c}}), \vec{\boldsymbol{h}}) \in \mathbb{X}_q^n$ and $\vec{\boldsymbol{t}}' = \vec{\boldsymbol{V}}'\vec{\boldsymbol{z}} + \vec{\boldsymbol{K}}'\vec{\boldsymbol{s}}' - \vec{\boldsymbol{b}} \in \mathbb{X}_{q'}^{n'}$. If $\vec{\boldsymbol{t}}$ is $\perp$, return 0.
- Check $\|\vec{\boldsymbol{z}}\| \overset{?}{\leq} \gamma$, $\|\vec{\boldsymbol{s}}\| \overset{?}{\leq} \gamma$ and $\|\vec{\boldsymbol{s}}'\| \overset{?}{\leq} B$. Otherwise, return 0.
- Return $\vec{x} \overset{?}{=} \texttt{hash}(\vec{\boldsymbol{c}}, \vec{\boldsymbol{b}}, \vec{\boldsymbol{t}}, \vec{\boldsymbol{t}}')$.

Security. This protocol is equal to Fiat-Shamir lattice signatures [40, 48] where we compute two signatures instead of one to prove the knowledge of v. Therefore, we claim this protocol is complete and knowledge sound when iMSIS of $(n', m'+L, q', B, N)$ is hard and Approx-SIS of $(n, m+1, q, \gamma, \gamma', N)$ is hard based on [40].

C Bullet-Proof Range Proofs

```
▷ BP [14] with strong Fiat Shamir challenges. Here, $H : \{0,1\}^* \rightarrow \mathbb{Z}_q^+$ and $\delta(y,z) = (z - z^2) \cdot \langle \mathbf{1}^l, \boldsymbol{y}^l \rangle - z^3 \langle \mathbf{1}^l, \mathbf{2}^l \rangle \in \mathbb{Z}_q^+$
BRP.Prv($pp, C, v, k$):
$\boldsymbol{a}_L \in \{0,1\}^l$ s.t. $\langle \boldsymbol{a}_L, \mathbf{2}^l \rangle = v$
$\boldsymbol{a}_R := \boldsymbol{a}_L - \mathbf{1}^l$
$\boldsymbol{s}_L, \boldsymbol{s}_R \xleftarrow{\$} (\mathbb{Z}_q^+)^l$
$\alpha, \rho \xleftarrow{\$} \mathbb{Z}_q^+$
$A \leftarrow g^\alpha \boldsymbol{h}^{\boldsymbol{a}_L} \boldsymbol{g}^{\boldsymbol{a}_R} \in \mathbb{G}$
$S \leftarrow g^\rho \boldsymbol{h}^{\boldsymbol{s}_L} \boldsymbol{g}^{\boldsymbol{s}_R} \in \mathbb{G}$
$y \leftarrow H(C, A, S),\ z \leftarrow H(A, S, y)$
$L(X) := (\boldsymbol{a}_L - z \cdot \mathbf{1}^l) + \boldsymbol{s}_L \cdot X$
$R(X) := \boldsymbol{y}^l \cdot (\boldsymbol{a}_R + z \cdot \mathbf{1}^l + \boldsymbol{s}_R \cdot X) + z^2 \cdot \mathbf{2}^l$
▷ computes $t(X) := \langle L(X), R(X) \rangle$
$t(X) := t_0 + t_1 \cdot X + t_2 \cdot X^2$
$\tau_1, \tau_2 \xleftarrow{\$} \mathbb{Z}_q^+$
$T_1 \leftarrow h^{t_1} g^{\tau_1}, T_2 \leftarrow h^{t_2} g^{\tau_2}$
$x \leftarrow H(A, S, y, z, T_1, T_2)$
$\boldsymbol{l} := L(x) = (\boldsymbol{a}_L - z \cdot \mathbf{1}^l) + \boldsymbol{s}_L \cdot x \in (\mathbb{Z}_q^+)^l$
$\boldsymbol{r} := R(x) = \boldsymbol{y}^l \cdot (\boldsymbol{a}_R + z \cdot \mathbf{1}^l + \boldsymbol{s}_R \cdot x) + z^2 \cdot \mathbf{2}^l \in (\mathbb{Z}_q^+)^l$
$\hat{t} := \langle \boldsymbol{l}, \boldsymbol{r} \rangle \in \mathbb{Z}_q^+$
$\tau_x := \tau_2 \cdot x^2 + \tau_1 \cdot x + z^2 \cdot k \in \mathbb{Z}_q^+$
$\mu := \alpha + \rho \cdot x$
$x_{\texttt{IP}} \leftarrow H(A, S, y, z, T_1, T_2, \tau_x, \mu, \hat{t}\ )$
$\pi_{\texttt{IP}}$ := IP.Prove($pp, \boldsymbol{h}, \boldsymbol{g}, u, x_{\texttt{IP}}, C, \boldsymbol{h}^{\boldsymbol{l}} \boldsymbol{g}^{\boldsymbol{r}}, \hat{t}, \boldsymbol{l}, \boldsymbol{r}$)
return $\pi := (A, S, T_1, T_2, \tau_x, \mu, \hat{t}, \pi_{\texttt{IP}})$
```

```
BRP.Ver($pp, C, \pi$): ▷ verification
$(A, S, T_1, T_2, \tau_x, \mu, \hat{t}, \pi_{\texttt{IP}}) := \pi$
$y := H(C, A, S),\ z := H(A, S, y),\ x := H(A, S, y, z, T_1, T_2)$
$\boldsymbol{g}' := \boldsymbol{g}^{(\boldsymbol{y}^{-l})}$ //$\boldsymbol{g}' = \{\boldsymbol{g}_1, \boldsymbol{g}_2^{y^{-1}}, \boldsymbol{g}_3^{y^{-2}}, .., \boldsymbol{g}_l^{y^{-l+1}}\}$
if $h^{\hat{t}} g^{\tau_x} \neq C^{z^2} \cdot h^{\delta(y,z)} \cdot T_1^x \cdot T_2^{x^2}$:
    return 0
$P = A \cdot S^x \cdot \boldsymbol{h}^{-z} \cdot (\boldsymbol{g}')^{z \cdot \boldsymbol{y}^l + z^2 \cdot \mathbf{2}^l}$
$x_{\texttt{IP}} \leftarrow H(A, S, y, z, T_1, T_2, \tau_x, \mu, \hat{t}\ )$
return IP.verify($pp, \boldsymbol{h}, \boldsymbol{g}', u, x_{\texttt{IP}}, C, Pg^{-\mu}, \hat{t}, \pi_{\texttt{IP}}$)

▷ Inner Product Argument - Prove
IP.Prove($pp, \boldsymbol{g}, \boldsymbol{h}, u, x_{\texttt{IP}}, C, P, c, \boldsymbol{a}, \boldsymbol{b}$)
$P' \leftarrow P \cdot u^{x_{\texttt{IP}} \cdot c}$
$(g, h, C, c, P, a, b, \boldsymbol{l}, \boldsymbol{r})$ := IP.Prove($pp$,
    $\boldsymbol{h}, \boldsymbol{g}, C, c, P', \boldsymbol{a}, \boldsymbol{b}, \{\}, \{\}$)
// Here $\boldsymbol{l}, \boldsymbol{r} \in \mathbb{G}^{\log_2 |a|}$
return $\pi_{\texttt{IP}} = (a, b, \boldsymbol{l}, \boldsymbol{r})$

▷ A recursive function
$\boldsymbol{a}_{[:n]} = \{\boldsymbol{a}_1, .., \boldsymbol{a}_{n-1}\}$
$\boldsymbol{a}_{[n:]} = \{\boldsymbol{a}_n, .., \boldsymbol{a}_{|\boldsymbol{a}|}\}$
IP.ComputeProof($pp, \boldsymbol{g}, \boldsymbol{h}, C, c, P, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{l}, \boldsymbol{r}$)
if $|\boldsymbol{g}| \neq |\boldsymbol{h}| \neq |\boldsymbol{a}| \neq |\boldsymbol{b}|$: return $\perp$
$n = |\boldsymbol{g}|$
if $n = 1$: return $(\boldsymbol{g}, \boldsymbol{h}, C, c, P, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{l}, \boldsymbol{r})$
else:
    $n' := n/2$
    $c_L \leftarrow \langle \boldsymbol{a}_{[:n']}, \boldsymbol{b}_{[n':]} \rangle \in \mathbb{Z}_q^+$
    $c_R \leftarrow \langle \boldsymbol{a}_{[n':]}, \boldsymbol{b}_{[:n']} \rangle \in \mathbb{Z}_q^+$
    $L \leftarrow \langle \boldsymbol{g}_{[n':]}^{\boldsymbol{a}_{[:n']}}, \boldsymbol{h}_{[:n']}^{\boldsymbol{b}_{[n':]}} \rangle \in \mathbb{G}$
    $R \leftarrow \langle \boldsymbol{g}_{[:n']}^{\boldsymbol{a}_{[n':]}}, \boldsymbol{h}_{[n':]}^{\boldsymbol{b}_{[:n']}} \rangle \in \mathbb{G}$
    ▷ add $L, R$ to $\boldsymbol{l}, \boldsymbol{r}$
    $\boldsymbol{l} := \boldsymbol{l} \| L$, $\boldsymbol{r} := \boldsymbol{r} \| R$
    $x \leftarrow H(C, L, R)$
    ▷ element-wise
    $\boldsymbol{g}' := \boldsymbol{g}_{[:n']}^{x^{-1}} \odot \boldsymbol{g}_{[n':]}^{x} \in \mathbb{G}^{n'}$
    $\boldsymbol{h}' := \boldsymbol{h}_{[:n']}^{x} \odot \boldsymbol{h}_{[n':]}^{x^{-1}} \in \mathbb{G}^{n'}$
    $P' := L^{x^2} P R^{-x^2} \in \mathbb{G}$
    $\boldsymbol{a'} = \boldsymbol{a}_{[:n']} x + \boldsymbol{a}_{[n':]} x^{-1} \in (\mathbb{Z}_q^+)^{n'}$
    $\boldsymbol{b'} = \boldsymbol{b}_{[:n']} x^{-1} + \boldsymbol{b}_{[n':]} x \in (\mathbb{Z}_q^+)^{n'}$
    $(\boldsymbol{g}, \boldsymbol{h}, C, c, P, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{l}, \boldsymbol{r}) := (\boldsymbol{g}', \boldsymbol{h}',$
      $C, c, P', \boldsymbol{a}', \boldsymbol{b}', \boldsymbol{l}, \boldsymbol{r})$
run recursively IP.ComputeProof($pp$,
    $\boldsymbol{g}, \boldsymbol{h}, C, c, P, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{l}, \boldsymbol{r}$)
```

```
▷ Inner Product Argument - Verify
IP.Verify($pp, \boldsymbol{g}, \boldsymbol{h}, u, x_{\mathtt{IP}}, C, P, c, \pi_{\mathtt{IP}}$)
$P \leftarrow P \cdot u^{x_{\mathtt{IP}} \cdot c}$
$(a, b, \boldsymbol{l}, \boldsymbol{r}) := \pi_{\mathtt{IP}}$
if $\log_2 |\boldsymbol{g}| \neq \log_2 |\boldsymbol{h}| \neq |\boldsymbol{l}| \neq |\boldsymbol{r}|$:
  return $\perp$
$n' = |\boldsymbol{g}|$
for $(L, R) \in (\boldsymbol{l}, \boldsymbol{r})$
  $n' := n'/2$
  $x \leftarrow H(C, L, R)$
  ▷ element-wise product $\odot$
  $\boldsymbol{g} := \boldsymbol{g}_{[:n']}^{x^{-1}} \odot \boldsymbol{g}_{[n':]}^{x} \in \mathbb{G}^{n'}$
  $\boldsymbol{h} := \boldsymbol{h}_{[:n']}^{x} \odot \boldsymbol{h}_{[n':]}^{x^{-1}} \in \mathbb{G}^{n'}$
  $P := L^{x^2} P R_i^{-x^2} \in \mathbb{G}$
▷ Note that $|\boldsymbol{g}| = 1 \longrightarrow \boldsymbol{g} = g$ and
$|\boldsymbol{h}| = 1 \longrightarrow \boldsymbol{h} = h$
return $P \overset{?}{=} g^a h^b u^{x_{\mathtt{IP}} \cdot a \cdot b}$
```